

原书第8版



本科教学版

软件工程

实践者的研究方法

[美] 罗杰 S. 普莱斯曼 (Roger S. Pressman) 著 郑人杰 马素霞 等译
布鲁斯 R. 马克西姆 (Bruce R. Maxim)

Software Engineering

A Practitioner's Approach Eighth Edition · Chinese Abridgement

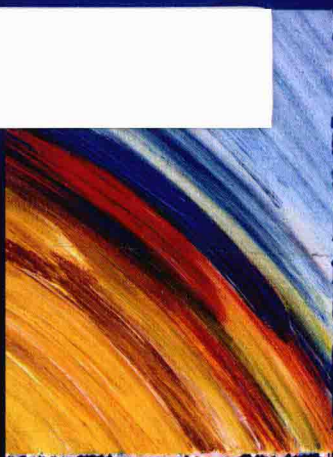
Eighth Edition

Software Engineering

A PRACTITIONER'S APPROACH

Roger S.
PRESSMAN

Bruce R.
MAXIM



机械工业出版社
China Machine Press

软件工程 实践者的研究方法 (原书第8版·本科教学版)

Software Engineering A Practitioner's Approach Eighth Edition · Chinese Abridgement

本书是软件工程领域的经典权威著作,自第1版出版至今,30多年来在软件工程界产生了巨大而深远的影响。第8版在继承之前版本风格与优势的基础上,不仅更新了全书内容,而且优化了篇章结构,以内容全面而著称。

为满足本科生“软件工程”课程的教学需求,本书保留了完整版中的基本内容,压缩或删除了一些高级内容,涵盖软件过程、建模、质量管理和项目管理四项主题,对概念、原则、方法和工具的介绍细致、清晰且实用。此外,书中提供了丰富的辅助阅读资源并配有网络资源(www.mhhe.com/pressman)。

本书特色

- 重点与更新。突出软件质量管理的相关内容,同时加强了软件过程部分。与时俱进的内容包括系统安全性和软件工程对人员的要求等。
- 组织与结构。每章开篇给出“要点浏览”和“关键概念”,最后给出“习题与思考题”以及“扩展阅读与信息资源”,章中贯穿的辅助阅读信息包括SafeHome对话框、软件工具、引述、关键点、网络资源等。
- 教学与自学。本科教学版更适合作为高校教材,同时,精选的知识点和丰富的扩展资源不仅易于初学者掌握基础理论,而且可供有兴趣的读者进行深入研究。

作者简介

罗杰 S. 普莱斯曼 (Roger S. Pressman) 软件工程界国际知名的顾问和作家,作为工程师、经理人、教授、演讲家和企业家奋战在这一领域已40余年。现任一家咨询公司的总裁,致力于协助企业建立有效的软件工程实践。还是一家创业公司的创始人,专注于为特斯拉 Model S 系列电动车生产定制产品。



布鲁斯 R. 马克西姆 (Bruce R. Maxim) 曾任软件工程师、项目经理、教授、作家和咨询师,研究兴趣涉及软件工程、人机交互、游戏设计及教育等领域,曾任某游戏开发公司的首席技术官。现任密歇根大学迪尔伯恩分校副教授,为该校工程与计算机科学学院建立了游戏实验室。



Mc
Graw
Hill
Education

www.mheducation.com

投稿热线: (010) 88379604
客服热线: (010) 88378991 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn

上架指导: 计算机软件工程

ISBN 978-7-111-55501-8



9 787111 555018 >

定价: 59.00元

封面设计: 邹逸 林杉

计 算 机 科 学 丛 书

原书第8版



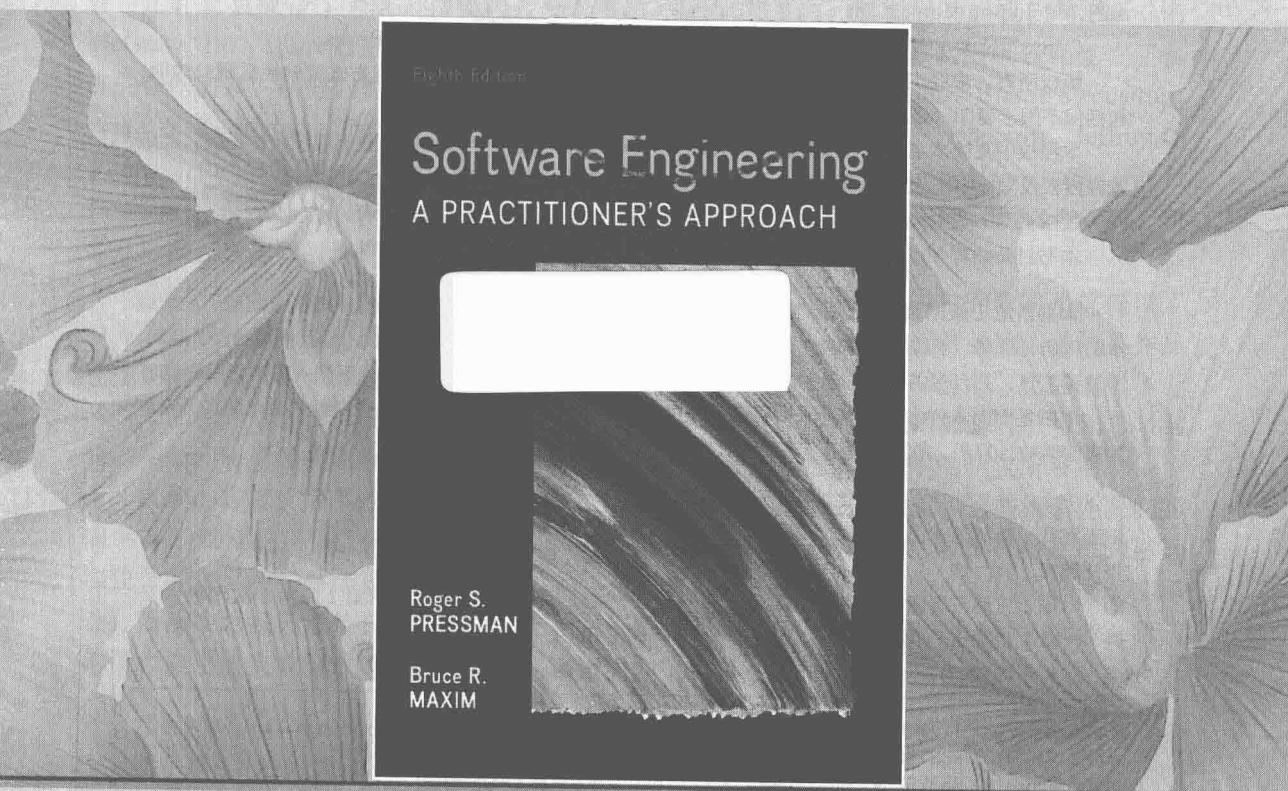
软件工程

实践者的研究方法

[美] 罗杰 S. 普莱斯曼 (Roger S. Pressman) 著 郑人杰 马素霞 等译
布鲁斯 R. 马克西姆 (Bruce R. Maxim)

Software Engineering

A Practitioner's Approach, Eighth Edition · Chinese Abridgement



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

软件工程: 实践者的研究方法 (原书第 8 版·本科教学版) / (美) 罗杰 S. 普莱斯曼 (Roger S. Pressman), (美) 布鲁斯 R. 马克西姆 (Bruce R. Maxim) 著; 郑人杰等译. —北京: 机械工业出版社, 2016.12

(计算机科学丛书)

书名原文: Software Engineering: A Practitioner's Approach, Eighth Edition

ISBN 978-7-111-55501-8

I. 软… II. ① 罗… ② 布… ③ 郑… III. 软件工程 - 高等学校 - 教材 IV. TP311.5

中国版本图书馆 CIP 数据核字 (2016) 第 294545 号

本书版权登记号: 图字: 01-2014-4759

Roger S. Pressman, Bruce R. Maxim: Software Engineering: A Practitioner's Approach, Eighth Edition (978-0-07-802212-8).

Copyright © 2015 by McGraw-Hill Education.

All Rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese abridgement is jointly published by McGraw-Hill Education and China Machine Press. This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2017 by McGraw-Hill Education and China Machine Press.

版权所有。未经出版人事先书面许可, 对本出版物的任何部分不得以任何方式或途径复制或传播, 包括但不限于复印、录制、录音, 或通过任何数据库、信息或可检索的系统。

本授权中文简体字删减版由麦格劳 - 希尔 (亚洲) 教育出版公司和机械工业出版社合作出版。此版本经授权仅限在中华人民共和国境内 (不包括香港、澳门特别行政区及台湾地区) 销售。

版权 © 2017 由麦格劳 - 希尔 (亚洲) 教育出版公司与机械工业出版社所有。

本书封面贴有 McGraw-Hill Education 公司防伪标签, 无标签者不得销售。

本书自第 1 版出版至今, 30 多年来在软件工程界产生了巨大而深远的影响。第 8 版继承了之前版本的风格与优势, 系统地讲解软件过程、建模、质量管理、项目管理等基础知识, 涵盖相关概念、原则、方法和工具, 并且提供丰富的辅助阅读资源和网络资源, 指导有兴趣的读者进行更深入的学习和研究。

本书是面向本科生的版本, 保留了完整版中的基础内容, 压缩或删除了一些高级内容, 更加适合作为高等院校计算机、软件工程及相关专业的软件工程课程教材。

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 曲 熠

责任校对: 殷 虹

印 刷: 三河市宏图印务有限公司

版 次: 2017 年 1 月第 1 版第 1 次印刷

开 本: 185mm × 260mm 1/16

印 张: 25.75

书 号: ISBN 978-7-111-55501-8

定 价: 59.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅肇划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum, Bjarne Stroustrup, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方式如下：

华章网站：www.hzbook.com

电子邮件：hzsj@hzbook.com

联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037



华章科技图书出版中心

改编者序

Software Engineering: A Practitioner's Approach, Eighth Edition, Chinese Abridgement

Pressman 博士的《软件工程：实践者的研究方法》(第 8 版)加入了 Maxim 博士的工作，一如既往是软件工程领域的经典教材。

此次本科教学版的推出，主要是为了与第 7 版本科教学版保持一致，使内容更符合普通高校本科生的课程大纲，适合本科学生在一学期内掌握软件工程最核心的基础内容，也帮助更多学生更好地接受英文原版教材。

与原版相比，第 8 版本科教学版的改动内容如下。

我们注意到新版增加了移动 App 和安全工程这两块非常重要的内容。然而，考虑到课时限制，我们只保留了安全工程部分，将移动 App 的内容留给研究生课程。与第 7 版的改动宗旨相似，我们主要关注公共框架中的基本概念，而将 WebApp、移动 App、度量及其他高级课题内容留给研究生阶段的课程。

本科教学版不包含但适合研究生阶段学习的章节有：完整版第 16 章（基于模式的设计），第 17 章（WebApp 设计），第 18 章（移动 App 设计），第 20 章（评审技术），第 25 章（测试 WebApp），第 26 章（测试移动 App），第 28 章（形式化建模与验证），第 36 章（维护与再工程），以及关于高级课题的第五部分。完整版第 30 章（产品度量）也是比较高级的课题，但概要介绍框架性概念的 30.1 节可保留，此节被移到第 21 章（软件质量保证）最后，变为 21.10 节。

此外，为了与第 7 版本科教学版保持一致，下列章节中还做了少量裁减：

1. 所有与 WebApp 和移动 App 相关的章节都被删除，具体包括：11.5 节、14.5 节、15.5 节、22.5 ~ 22.6 节、29.4 节、32.2.6 节、34.5.4 节。
2. 第 3、4 章：裁减 3.5 节和 4.4 ~ 4.5 节属于研究生课程中的高级内容。
3. 第 7 章：该章只是引言，其核心内容将在后面的章节中逐一展开介绍。为减少学生必须阅读的篇幅，该章被整体删除。
4. 第 8 章：裁减 8.2.5 ~ 8.2.6 节、8.6 ~ 8.8 节属于研究生课程中的高级内容。
5. 第 23 章：裁减 23.4.4 节、23.6.1 节、23.6.4 节、23.8 ~ 23.10 节属于研究生课程中的高级内容。
6. 对于本科课程而言，了解关于度量和估算的基本概念就足够了，因此 32.4 ~ 32.6 节以及 33.9 ~ 33.10 节也作为高级内容被裁减。

经过压缩，英文完整版 900 余页的篇幅被大幅压缩到中文版 400 页左右。由于我们只是就知识点做了裁减，对于保留部分并未删减原作的语言和案例，所以不会对写作风格造成明显的破坏。鉴于改编者的经验和学识有限，对于裁减内容的定夺或存在欠妥之处，欢迎广大读者批评指正。

陈越

2016 年 11 月

本书是国际知名软件工程专家罗杰 S. 普莱斯曼 (Roger S. Pressman) 的最新著作。自 35 年前第 1 版问世以来, 这本书在软件工程界产生了巨大而深远的影响。其权威性是无可置疑的, 在培养软件工程专业人才方面所起的作用也是显而易见的。在这一版中, 新加入的布鲁斯 (Bruce) 作为第二作者参与了本书的编写工作。

我自 20 世纪 80 年代中期开始从事高校软件工程方面的教学与科研工作, 多年来, 这本书的各个版本一直是我的重要参考书, 它给了我许多启发和帮助, 我也曾多次向许多业界好友和学生推荐此书。

如今基于计算机的系统已经广泛而深入地渗透到经济、国防和人们日常生活的各个领域, 特别是在互联网的推动下, 不仅许多行业得以改进和更新, 而且产生了一批新的行业, 展现了全新的业态。我们必须意识到, 在计算机不断向社会的深度和广度层面发展的过程中, 软件始终处在系统的核心地位, 起着中枢和灵魂的作用, 而且这种作用正日益突出。因此, 如何为现代化系统配备合格和优良的软件也就更加受到人们的广泛关注。

本书系统地论证了软件工程领域的基本知识和最新研究成果, 包括新的概念、原则、技术、方法和工具。同时书中还为读者提供了进一步学习和研究的线索, 包括许多可供利用的网上资料和信息。与第 7 版相比, 本版继承了一些优点, 同时也做了一些改动、扩充和更新。

本书特点

1. 全书内容分为四个部分, 共 26 章, 还包括两个附录。四个部分的内容分别为软件过程、建模、质量管理和管理软件项目。
2. 本书继承了前一版的特色, 突出了软件质量管理的内容, 同时也加强了软件过程部分。此外, 增加的内容还包括: 软件工作对人员的要求; 近年来软件产业发展中出现的新课题——软件系统安全性和移动 App。
3. 仍然在各章的开头给出“要点浏览”(包括概念、人员、重要性、步骤、工作产品和质量保证措施) 以及“关键概念”(全章内容的关键词)。
4. 仍然在各章的末尾给出“习题与思考题”以及“扩展阅读与信息资源”, 这些都非常适合有兴趣、有需要的读者沿着所提供的线索开展进一步的学习和研究。
5. 仍然保留了本书历次版本在各章中为读者提供的多种形式的辅助阅读信息, 可以说这是本书的一个突出特点。这些信息从形式上分为两类: 一类是采用通栏形式的说明框, 包括要点浏览、信息栏、软件工具和 SafeHome 对话框等; 另一类是居于页面右侧的说明框, 包括关键概念、引述、建议、关键点、提问和网络资源。这些说明框非常有益于读者理解和进一步探索相关内容。

读者对象

本书仍然面向三类读者, 即高校学生 (特别是研究生)、教师和专业软件技术人员。总

体而言，本书适合作为高校计算机或信息技术相关专业的教学用书，特别适合为软件工程课程提供教学服务。

译者说明

参与本书翻译工作的译者以华北电力大学和清华大学的教师为主，也有少数软件企业和中国软件行业协会的研究人员。他们是：马素霞（第1～4章、17～19章及附录1～2）、宋兰（第7～10章及21章）、石敏（第11、12章）、周长玉（第13、14章）、韩新启（第15～16章）、王海青（第22、26章）、王素琴（第22～25章）。此外，刘瑾完成了第5、6章的翻译工作。我负责第20章以及前言和作者简介部分。在翻译过程中，我们得到了华北电力大学控制与计算机工程学院洪海、熊里、赵敏、李树超、高晶晶、吕骁同学的帮助，在此对他们的辛勤劳动表示感谢。我对全部译稿、马素霞教授对大部分译稿做了仔细审核与修改，并更正了原书中个别的错漏之处。

本书英文完整版有900多页，翻译工作量巨大，而译者均有繁重的本职工作，时间并不宽松，因此译文中难免有不当之处，敬请读者见谅并不吝指正。

总之，这是一本非常优秀的软件工程读物，本人十分高兴地向国内读者推荐。我们相信，认真阅读它，定会使你获益匪浅。

郑人杰

2016年11月

如果有这样一款计算机软件，它能满足用户的需求，能在相当长的时间内无故障地运行，修改起来轻松便捷，使用起来更是得心应手，那么，这款软件必定是成功的，它切实改善了我们的生活。但是，如果有这样一款软件，它令用户失望，错误频出，修改起来困难重重，使用起来更是举步维艰，那么，这必定是一款失败的软件，它使我们的生活一团糟。谁都希望开发出优秀的软件，为我们的生活带来便利，而不是把自己陷入失败的深渊。要想使软件获得成功，在设计和构建软件时就需要有规范，需要采用工程化的方法。

自本书第1版问世以来的近35年中，软件工程已经从少数倡导者提出的一些朦胧概念发展成为一门正规的工程学科，已被公认为是一个值得深入研究、认真学习和热烈讨论的课题。在整个行业中，软件工程师已经代替程序员成为人们优先选择的工作岗位，软件过程模型、软件工程方法和软件工具都已在全行业的所有环节成功采用。

尽管管理人员和一线专业人员都承认需要有更为规范的软件方法，但他们却始终在争论应该采用什么样的规范。有许多个人和公司至今仍在杂乱无章地开发着自己的软件，甚至即使他们正在开发的系统要服务于当今最为先进的技术，状况也仍是如此。许多专业人员和学生并不了解现代方法，这导致他们所开发的软件质量很差，因而造成了严重的后果。此外，有关软件工程方法真实本质的争论一直持续进行着。软件工程的地位问题已成为一门对比研究课题。人们对软件工程的态​​度已经有所改善，研究工作已取得了进展，不过要成为一门完全成熟的学科，我们还有大量的工作要做。

我们希望本书能够成为引导读者进入正在走向成熟的软件工程学科的入门读物，和以前的7个版本一样，第8版对学生和专业人员同样具有很强的吸引力。它既是软件专业人员的工作指南，也是高年级本科生和一年级研究生的综合性参考书。

第8版中包含了许多新的内容，它绝不只是前一版的简单更新。这一版不仅对内容做了适当的修改，而且调整了全书的结构，以改进教学顺序；同时更加强调一些新的和重要的软件工程过程和软件工程实践知识。此外，本书进一步加强了“支持系统”，为学生、教师和专业人员提供了更为丰富的知识资源。读者可访问专门为本书建立的网站（www.mhhe.com/pressman）查阅这些信息。

篇章结构

本书共26章，分为四个部分。这种划分有利于那些无法在一个学期内讲完全书内容的教师灵活安排教学。

第一部分“软件过程”给出了有关软件过程的各种不同观点，讨论了所有重要的过程模型，还涉及惯用过程和敏捷过程在指导思想上的分歧。第二部分“建模”给出了分析方法和设计方法，重点讲解面向对象方法和UML建模。第三部分“质量管理”介绍了有关质量管理的概念、规程和方法，使得软件团队能够很好地评估软件质量，实施软件质量保证规程，并正确地运用有效的测试策略和战术。第四部分“管理软件项目”介绍了与计划、管理和控制软件开发项目的人员有关的问题。

第8版沿用了前面几个版本的做法，在各章中都提供了大量的辅助阅读信息，包括一个虚拟软件团队在工作中遇到困难时展开的对话，还包括对各章相关知识给出的补充方法和工具。

致谢

我们要特别感谢渥太华大学的 Tim Lethbridge，他帮助我们开发了 UML 和 OCL 的案例，以及配合本书内容的其他案例研究。Colby 学院的 Dale Skrien 开发了附录 1 的 UML 教辅资源。他们的帮助和意见都是十分宝贵的。此外也感谢高级软件工程师 Austin Krauss，他提供了关于视频游戏产业软件开发的宝贵意见。同时，要对为第8版评审做出贡献的几位教授表示感谢，他们是佛罗里达大学的 Manuel E. Bermudez、堪萨斯州立大学的 Scott DeLoach、密歇根州立大学的 Alex Liu 和犹他州立大学的 Dean Mathias。正是他们的详尽而真诚的评审意见帮助我们，使得本书更加完善。

特别感谢

十分高兴有机会与罗杰合作，参与本书第8版的撰写工作。在此期间我的儿子 Benjamin 推出了他的第一款移动 App，我的女儿 Katherine 开始了她的室内设计生涯。我十分高兴地看到他们已经长大成人。同时非常感谢妻子 Norma，她热情地支持我，使我能够将所有空闲时间都投入本书的写作。

布鲁斯 R. 马克西姆 (Bruce R. Maxim)

随着本书各版本的不断推出，我的两个儿子 Mathew 和 Michael 也逐渐从小男孩成长为男子汉。他们在生活中的成熟、品格和成功鼓舞着我，没有什么比这更让我自豪了。他们现在也已经有了自己的孩子——Maya 和 Lily，这两个女孩已经是移动计算时代新智能设备方面的奇才。最后要感谢妻子 Barbara，她宽容我花费如此多的时间在办公室工作，并且还鼓励我继续写作本书的下一个版本。

罗杰 S. 普莱斯曼 (Roger S. Pressman)

出版者的话

改编者序

译者序

前言

第 1 章 软件的本质 1

1.1 软件的本质 3

1.1.1 定义软件 3

1.1.2 软件应用领域 4

1.1.3 遗留软件 5

1.2 软件的变更本质 6

1.2.1 WebApp 6

1.2.2 移动 App 7

1.2.3 云计算 7

1.2.4 产品线软件 8

习题与思考题 8

扩展阅读与信息资源 8

第 2 章 软件工程 10

2.1 定义软件工程学科 11

2.2 软件过程 11

2.2.1 过程框架 12

2.2.2 普适性活动 12

2.2.3 过程的适应性调整 13

2.3 软件工程实践 13

2.3.1 实践的精髓 14

2.3.2 通用原则 14

2.4 软件开发神话 16

2.5 这一切是如何开始的 18

习题与思考题 19

扩展阅读与信息资源 19

第一部分 软件过程

第 3 章 软件过程结构 22

3.1 通用过程模型 23

3.2 定义框架活动 24

3.3 明确任务集 24

3.4 过程模式 25

习题与思考题 27

扩展阅读与信息资源 27

第 4 章 过程模型 29

4.1 惯用过程模型 30

4.1.1 瀑布模型 30

4.1.2 增量过程模型 32

4.1.3 演化过程模型 32

4.1.4 并发模型 36

4.1.5 演化过程的最终评述 37

4.2 专用过程模型 38

4.2.1 基于构件的开发 38

4.2.2 形式化方法模型 39

4.2.3 面向方面的软件开发 39

4.3 统一过程 40

4.3.1 统一过程的简史 41

4.3.2 统一过程的阶段 41

4.4 产品和过程 42

习题与思考题 43

扩展阅读与信息资源 43

第 5 章 敏捷开发 45

5.1 什么是敏捷 46

5.2 敏捷及变更成本 47

5.3 什么是敏捷过程 47

5.3.1 敏捷原则 48

5.3.2 敏捷开发战略	49	7.3.2 质量功能部署	82
5.4 极限编程	49	7.3.3 使用场景	83
5.4.1 极限编程过程	49	7.3.4 获取工作产品	84
5.4.2 工业极限编程	51	7.3.5 敏捷需求获取	84
5.5 其他敏捷过程模型	53	7.3.6 面向服务的方法	84
5.5.1 Scrum	53	7.4 开发用例	85
5.5.2 动态系统开发方法	54	7.5 构建分析模型	88
5.5.3 敏捷建模	55	7.5.1 分析模型的元素	89
5.5.4 敏捷统一过程	56	7.5.2 分析模式	91
5.6 敏捷过程工具集	57	7.5.3 敏捷需求工程	91
习题与思考题	58	7.5.4 自适应系统的需求	91
扩展阅读与信息资源	58	7.6 避免常见错误	92
第 6 章 软件工程的人员方面	60	习题与思考题	92
6.1 软件工程师的特质	60	扩展阅读与信息资源	93
6.2 软件工程心理学	61	第 8 章 需求建模：基于场景的方法	95
6.3 软件团队	62	8.1 需求分析	96
6.4 团队结构	63	8.1.1 总体目标和原理	96
6.5 敏捷团队	64	8.1.2 分析的经验原则	97
6.5.1 通用敏捷团队	64	8.1.3 域分析	97
6.5.2 XP 团队	65	8.1.4 需求建模的方法	99
6.6 社交媒体的影响	66	8.2 基于场景建模	100
6.7 软件工程中云的应用	67	8.2.1 创建初始用例	100
6.8 协作工具	67	8.2.2 细化初始用例	102
6.9 全球化团队	68	8.2.3 编写正式用例	103
习题与思考题	69	8.3 补充用例的 UML 模型	105
扩展阅读与信息资源	69	8.3.1 开发活动图	105
第二部分 建模		8.3.2 泳道图	106
第 7 章 理解需求	72	习题与思考题	107
7.1 需求工程	73	扩展阅读与信息资源	107
7.2 建立根基	78	第 9 章 需求建模：基于类的方法	108
7.2.1 确认利益相关者	78	9.1 识别分析类	108
7.2.2 识别多重观点	78	9.2 描述属性	111
7.2.3 协同合作	79	9.3 定义操作	111
7.2.4 首次提问	79	9.4 类-职责-协作者建模	113
7.3 获取需求	80	9.5 关联和依赖	118
7.3.1 协作收集需求	80	9.6 分析包	118
		习题与思考题	119

扩展阅读与信息资源	119	11.4.5 部署级设计元素	149
第 10 章 需求建模：行为和模式	121	习题与思考题	149
10.1 生成行为模型	121	扩展阅读与信息资源	150
10.2 识别用例事件	122	第 12 章 体系结构设计	152
10.3 状态表达	122	12.1 软件体系结构	153
10.4 需求建模的模式	125	12.1.1 什么是体系结构	153
10.4.1 发现分析模式	125	12.1.2 体系结构为什么重要	154
10.4.2 需求模式举例：执行器 - 传感器	126	12.1.3 体系结构描述	154
习题与思考题	129	12.1.4 体系结构决策	155
扩展阅读与信息资源	129	12.2 体系结构类型	156
第 11 章 设计概念	131	12.3 体系结构风格	156
11.1 软件工程中的设计	132	12.3.1 体系结构风格的简单 分类	157
11.2 设计过程	134	12.3.2 体系结构模式	159
11.2.1 软件质量指导原则和 属性	134	12.3.3 组织和求精	160
11.2.2 软件设计的演化	136	12.4 体系结构考虑要素	160
11.3 设计概念	137	12.5 体系结构决策	162
11.3.1 抽象	137	12.6 体系结构设计	162
11.3.2 体系结构	137	12.6.1 系统环境的表示	163
11.3.3 模式	138	12.6.2 定义原型	163
11.3.4 关注点分离	138	12.6.3 将体系结构细化为构件	164
11.3.5 模块化	138	12.6.4 描述系统实例	165
11.3.6 信息隐蔽	139	12.6.5 WebApp 的体系结构 设计	166
11.3.7 功能独立	139	13.6.6 移动 App 的体系结构 设计	166
11.3.8 求精	140	12.7 评估候选的体系结构设计	167
11.3.9 方面	140	12.7.1 体系结构描述语言	168
11.3.10 重构	141	12.7.2 体系结构评审	169
11.3.11 面向对象的设计概念	141	12.8 经验学习	169
11.3.12 设计类	142	12.9 基于模式的体系结构评审	170
11.3.13 依赖倒置	144	12.10 体系结构一致性检查	171
11.3.14 测试设计	145	12.11 敏捷性与体系结构	171
11.4 设计模型	145	习题与思考题	172
11.4.1 数据设计元素	146	扩展阅读与信息资源	173
11.4.2 体系结构设计元素	146	第 13 章 构件级设计	175
11.4.3 接口设计元素	147	13.1 什么是构件	176
11.4.4 构件级设计元素	148		

13.1.1	面向对象的观点	176
13.1.2	传统的观点	177
13.1.3	过程相关的观点	179
13.2	设计基于类的构件	180
13.2.1	基本设计原则	180
13.2.2	构件级设计指导方针	182
13.2.3	内聚性	183
13.2.4	耦合性	184
13.3	实施构件级设计	185
13.4	WebApp 的构件级设计	190
13.4.1	构件级内容设计	190
13.4.2	构件级功能设计	190
13.5	设计传统构件	190
13.6	基于构件的开发	191
13.6.1	领域工程	191
13.6.2	构件的合格性检验、适应性 修改与组合	191
13.6.3	体系结构不匹配	193
13.6.4	复用的分析与设计	193
13.6.5	构件的分类与检索	194
	习题与思考题	195
	扩展阅读与信息资源	195
第 14 章	用户界面设计	197
14.1	黄金规则	198
14.1.1	把控制权交给用户	198
14.1.2	减轻用户的记忆负担	199
14.1.3	保持界面一致	200
14.2	用户界面的分析和设计	201
14.2.1	用户界面分析和设计 模型	201
14.2.2	过程	202
14.3	界面分析	203
14.3.1	用户分析	203
14.3.2	任务分析和建模	204
14.3.3	显示内容分析	207
14.3.4	工作环境分析	207
14.4	界面设计步骤	208
14.4.1	应用界面设计步骤	208

14.4.2	用户界面设计模式	210
14.4.3	设计问题	210
14.5	设计评估	212
	习题与思考题	213
	扩展阅读与信息资源	214

第三部分 质量管理

第 15 章	质量概念	216
15.1	什么是质量	217
15.2	软件质量	218
15.2.1	Garvin 的质量维度	218
15.2.2	McCall 的质量因素	219
15.2.3	ISO 9126 质量因素	220
15.2.4	定向质量因素	220
15.2.5	过渡到量化观点	221
15.3	软件质量困境	222
15.3.1	“足够好”的软件	222
15.3.2	质量的成本	223
15.3.3	风险	225
15.3.4	疏忽和责任	225
15.3.5	质量和安全	225
15.3.6	管理活动的影响	226
15.4	实现软件质量	226
15.4.1	软件工程方法	227
15.4.2	项目管理技术	227
15.4.3	质量控制	227
15.4.4	质量保证	227
	习题与思考题	227
	扩展阅读与信息资源	228
第 16 章	软件质量保证	229
16.1	背景问题	230
16.2	软件质量保证的要素	230
16.3	软件质量保证的过程和产品 特性	232
16.4	软件质量保证的任务、目标和 度量	232
16.4.1	软件质量保证的任务	232

16.4.2 目标、属性和度量	233	17.6.1 恢复测试	260
16.5 软件质量保证的形式化方法	234	17.6.2 安全测试	260
16.6 统计软件质量保证	235	17.6.3 压力测试	260
16.6.1 一个普通的例子	235	17.6.4 性能测试	261
16.6.2 软件工程中的六西格玛	236	17.6.5 部署测试	261
16.7 软件可靠性	237	17.7 调试技巧	262
16.7.1 可靠性和可用性的测量	237	17.7.1 调试过程	262
16.7.2 软件安全	238	17.7.2 心理因素	263
16.8 ISO 9000 质量标准	239	17.7.3 调试策略	264
16.9 软件质量保证计划	240	17.7.4 纠正错误	265
16.10 产品度量框架	240	习题与思考题	265
16.10.1 测度、度量和指标	241	扩展阅读与信息资源	266
16.10.2 产品度量的挑战	241		
16.10.3 测量原则	242	第 18 章 测试传统的应用软件	268
16.10.4 面向目标的软件测量	242	18.1 软件测试基础	269
16.10.5 有效软件度量的属性	243	18.2 测试的内部视角和外部视角	270
习题与思考题	244	18.3 白盒测试	271
扩展阅读与信息资源	244	18.4 基本路径测试	271
第 17 章 软件测试策略	246	18.4.1 流图表示	271
17.1 软件测试的策略性方法	247	18.4.2 独立程序路径	273
17.1.1 验证与确认	247	18.4.3 生成测试用例	274
17.1.2 软件测试组织	248	18.5 控制结构测试	276
17.1.3 软件测试策略——宏观	249	18.6 黑盒测试	277
17.1.4 测试完成的标准	250	18.6.1 等价类划分	277
17.2 策略问题	251	18.6.2 边界值分析	278
17.3 传统软件的测试策略	251	18.7 基于模型的测试	278
17.3.1 单元测试	251	习题与思考题	279
17.3.2 集成测试	253	扩展阅读与信息资源	279
17.4 面向对象软件的测试策略	257		
17.4.1 面向对象环境中的单元 测试	257	第 19 章 测试面向对象的应用	281
17.4.2 面向对象环境中的集成 测试	257	19.1 扩展测试的视野	282
17.5 确认测试	258	19.2 测试 OOA 和 OOD 模型	282
17.5.1 确认测试准则	258	19.2.1 OOA 和 OOD 模型的 正确性	283
17.5.2 配置评审	258	19.2.2 面向对象模型的一致性	283
17.5.3 α 测试和 β 测试	258	19.3 面向对象测试策略	284
17.6 系统测试	260	19.3.1 面向对象环境中的单元 测试	284
		19.3.2 面向对象环境中的集成 测试	285

19.3.3 面向对象环境中的确认
测试 285

19.4 面向对象测试方法 285

19.4.1 面向对象概念的测试用例
设计含义 286

19.4.2 传统测试用例设计方法的
可应用性 286

19.4.3 基于故障的测试 286

19.4.4 基于场景的测试设计 287

19.5 类级可应用的测试方法 287

19.5.1 面向对象类的随机测试 287

19.5.2 类级的划分测试 288

19.6 类间测试用例设计 289

19.6.1 多类测试 289

19.6.2 从行为模型导出的测试 290

习题与思考题 291

扩展阅读与信息资源 291

第 20 章 安全性工程 293

20.1 安全性需求分析 294

20.2 网络世界中的安全性与
保密性 295

20.2.1 社交媒体 295

20.2.2 移动 App 296

20.2.3 云计算 296

20.2.4 物联网 296

20.3 安全性工程分析 296

20.3.1 安全性需求获取 297

20.3.2 安全性建模 297

20.3.3 测度设计 298

20.3.4 正确性检查 298

20.4 安全性保证 299

20.4.1 安全性保证过程 299

20.4.2 组织和管理 300

20.5 安全性风险分析 300

20.6 传统软件工程活动的作用 302

20.7 可信性系统验证 303

习题与思考题 304

扩展阅读与信息资源 305

第 21 章 软件配置管理 306

21.1 软件配置管理概述 307

21.1.1 SCM 场景 307

21.1.2 配置管理系统的元素 308

21.1.3 基线 309

21.1.4 软件配置项 310

21.1.5 依赖性和变更管理 310

21.2 SCM 中心存储库 311

21.2.1 一般特征和内容 311

21.2.2 SCM 特征 312

21.3 SCM 过程 312

21.3.1 软件配置中的对象标识 313

21.3.2 版本控制 314

21.3.3 变更控制 315

21.3.4 影响管理 317

21.3.5 配置审核 318

21.3.6 状态报告 318

习题与思考题 319

扩展阅读与信息资源 319

第四部分 管理软件项目

第 22 章 项目管理概念 322

22.1 管理涉及的范围 323

22.1.1 人员 323

22.1.2 产品 323

22.1.3 过程 324

22.1.4 项目 324

22.2 人员 324

22.2.1 利益相关者 324

22.2.2 团队负责人 325

22.2.3 软件团队 325

22.2.4 敏捷团队 327

22.2.5 协调和沟通问题 328

22.3 产品 329

22.3.1 软件范围 329

22.3.2 问题分解 329

22.4 过程 330

22.4.1 合并产品和过程 330

22.4.2 过程分解	330	24.6.4 基于 FP 估算的实例	358
22.5 项目	331	24.6.5 基于过程的估算	359
22.6 W ⁵ HH 原则	332	24.6.6 基于过程估算的实例	360
22.7 关键实践	333	24.6.7 基于用例的估算	360
习题与思考题	333	24.6.8 基于用例点估算的实例	361
扩展阅读与信息资源	334	24.6.9 调和不同的估算方法	362
第 23 章 过程度量与项目度量	336	24.7 经验估算模型	363
23.1 过程领域和项目领域中的 度量	337	24.7.1 估算模型的结构	363
23.1.1 过程度量和软件过程 改进	337	24.7.2 COCOMO II 模型	363
23.1.2 项目度量	339	24.7.3 软件方程	363
23.2 软件测量	340	24.8 面向对象项目的估算	364
23.2.1 面向规模的度量	341	习题与思考题	365
23.2.2 面向功能的度量	342	扩展阅读与信息资源	365
23.2.3 调和代码行度量和功能点 度量	342	第 25 章 项目进度安排	366
23.2.4 面向对象的度量	343	25.1 基本概念	367
23.2.5 面向用例的度量	344	25.2 项目进度安排概述	368
23.3 软件质量的度量	345	25.2.1 基本原则	369
23.3.1 测量质量	345	25.2.2 人员与工作量之间的 关系	370
23.3.2 缺陷排除效率	346	25.2.3 工作量分配	371
习题与思考题	347	25.3 为软件项目定义任务集	372
扩展阅读与信息资源	348	25.3.1 任务集举例	372
第 24 章 软件项目估算	350	25.3.2 主要任务的细化	373
24.1 对估算的观察	351	25.4 定义任务网络	373
24.2 项目计划过程	352	25.5 进度安排	374
24.3 软件范围和可行性	352	25.5.1 时序图	375
24.4 资源	353	25.5.2 跟踪进度	375
24.4.1 人力资源	354	25.5.3 跟踪面向对象项目的 进展	377
24.4.2 可复用软件资源	354	25.6 挣值分析	378
24.4.3 环境资源	354	习题与思考题	379
24.5 软件项目估算	354	扩展阅读与信息资源	380
24.6 分解技术	355	第 26 章 风险管理	382
24.6.1 软件规模估算	355	26.1 被动风险策略和主动风险 策略	383
24.6.2 基于问题的估算	356	26.2 软件风险	383
24.6.3 基于 LOC 估算的实例	357	26.3 风险识别	384

26.3.1 评估整体项目风险 385

26.3.2 风险因素和驱动因子 386

26.4 风险预测 387

26.4.1 建立风险表 387

26.4.2 评估风险影响 388

26.5 风险细化 390

26.6 风险缓解、监测和管理 390

26.7 RMMM 计划 392

习题与思考题 393

扩展阅读与信息资源 394

在线资源[⊖]

附录 1 UML 简介

附录 2 面向对象概念

参考文献

⊖ 请访问华章网站 (www.hzbook.com) 下载在线资源。

软件的本质

要点浏览

概念：计算机软件是由专业人员开发并长期维护的软件产品。完整的软件产品包括：可以在各种不同容量及系统结构的计算机上运行的程序、程序运行过程中产生的各种结果以及各种描述信息，这些信息可以以硬拷贝或是各种电子媒介形式存在。

人员：软件工程师开发软件并提供技术支持，产业界中几乎每个人都间接或直接地使用软件。

重要性：软件之所以重要是因为它在我们的生活中无所不在，并且日渐深入到商业、文化和日常生活的各个方面。

步骤：客户和利益相关者表达对计算机软件的要求，工程师构建软件产品，最终用户应用软件来解决特定的问题或者满足特定的要求。

工作产品：在一种或多种特定环境中运行并服务于一个或多个最终用户要求的计算机软件。

质量保证措施：如果你是软件工程师，就要应用本书中包含的思想。如果你是最终用户，应确保理解了自己的要求和环境，然后选择能很好地满足两者的应用软件。

在给我演示了最新开发的世界上最流行的第一人称射击视频游戏之后，这位年轻的开发者笑了。

“你不是一个玩家，对吗？”他问道。

我微笑道：“你是怎么猜到的？”

这位年轻人身着短裤和T恤衫。他的腿像活塞那样上下跳动，燃烧着神经能量，这在他的同事中看起来是很平常的。

“因为如果你是玩家，”他说，“你应该会更加兴奋。你已经看到了我们的下一代产品，我们的客户会对它着迷……这不是开玩笑。”

我们坐在开发区，这是地球上最成功的游戏开发者之一。在过去几年中，他演示的前几代游戏售出了5000万份，收入达几亿美元。

“那么，这一版什么时候上市？”我问道。

他耸耸肩，“大约在5个月以内，我们还有很多工作要做。”

他负责一个应用软件中的游戏和人工智能功能，该软件包含的代码超过了300万行。

“你们使用任何软件工程技术吗？”我问道，估计他会笑笑并摇头。

他停顿了一下，想了一会儿，然后缓慢地点点头。“我们让软件工程技术适应我们的需求，但是，我们确实使用。”

“在什么地方使用？”我试探地问道。

关键概念

应用领域

云计算

失效曲线

遗留软件

移动 App

产品线

软件定义

软件问题

软件的本质

磨损

WebApp

“我们的问题是经常将需求翻译成创意。”

“创意？”我打断了他的话。

“你知道，那些设计故事、人物及所有游戏素材的家伙，他们想的是游戏大卖。而我们不得不接受他们抛给我们的这些，并形成一组技术需求，从而构建游戏。”

“那么形成了需求之后呢？”

他耸耸肩，“我们不得不扩展并修改以前游戏版本的体系结构，并创建新的产品。我们需要根据需求创建代码，对每日构建的代码实施测试，并且做你书中建议的很多事情。”

“你知道我的书？”老实说，我非常惊讶。

“当然，在学校使用。那里有很多。”

“我已经与你的很多同事谈了，他们对我书中的很多东西持怀疑态度。”

他皱了皱眉，“你看，我们不是IT部门或航空公司，所以我们不得不对你所提倡的东西进行取舍。但是底线是一样的——我们需要生产高质量的产品，并且以可重复方式完成这一目标的唯一途径是改写我们自己的软件工程技术子集。”

“那么这个子集是如何随着时间的推移变更的？”

他停顿了一下，像是在思考着未来。“游戏将变得更加庞大和复杂，那是肯定的。随着更多竞争的出现，我们的开发时间表将会收缩。慢慢地，游戏本身会迫使我们应用更多的开发规范。如果我们不这样做，我们会死掉。”

计算机软件仍然是世界舞台上最为重要的技术，并且也是“意外效应法则”的典型例子。60年前，没有人曾预料到软件科学会成为今天商业、科学和工程所必需的技术。软件促进了新科技的创新（例如基因工程和纳米科技）、现代科技的发展（例如通信）以及传统技术的根本转变（例如媒体行业），软件技术已经成为个人计算机革命的推动力量，消费者使用智能手机就可以购买软件产品。软件还将由产品逐渐演化为服务，软件公司随需应变，通过Web浏览器发布即时更新功能，软件公司几乎可以比所有工业时代的公司都更大、更有影响力。在大量应用软件的驱动下，互联网将迅速发展，并逐渐改变人们生活的诸多方面——从图书馆搜索、消费购物、政治论战到年轻人和（不很年轻的）成年人的约会行为。

没有人曾想到软件可嵌入各种系统中：交通运输、医疗、通信、军事、工业、娱乐以及办公设备……不胜枚举。如果笃信“意外效应法则”的话，那么还有很多结果和影响是我们尚未预料到的。

没有人曾想到，随着时间的推移，将有数百万的计算机程序需要进行纠错、适应性调整和优化，这些维护工作将耗费比开发新软件更多的人力和物力。

随着软件重要性的日渐凸现，软件业界一直试图开发新的技术，使得高质量计算机程序的开发和维护更容易、更快捷、成本更低廉。一些技术主要针对特定应用领域（例如网站设计和实现），另一些着眼于技术领域（例如面向对象系统、面向方面的程序设计），还有一些覆盖面很宽（例如像Linux这样的操作系统）。然而，我们尚未开发出一种可以实现上述所有需求的软件技术，而且未来能够产生这种技术的可能性也很小。人们也尚未将其工作、享受、安全、娱乐、决策以及全部生活都完全依赖于计算机软件。这或许是正确的选择。

本书为需要构建正确软件的计算机软件工程师提供了一个框架。该框架包括过程、一系列方法以及我们称为软件工程的工具。

引述 创新观念和科技发现是经济增长的推进器。
——华尔街日报

1.1 软件的本质

现在的软件具有产品和产品交付载体的双重作用。作为产品，软件显示了由计算机硬件体现的计算能力，更广泛地说，显示的是由一个可被本地硬件设备访问的计算机网络体现的计算潜力。无论是安装在移动电话、手持平板电脑、台式机还是大型计算机中，软件都扮演着信息转换的角色：产生、管理、获取、修改、显示或者传输各种不同的信息，简单如几个比特的传递，复杂如从多个独立的数据源获取的多媒体演示。而作为产品生产的载体，软件提供了计算机控制（操作系统）、信息通信（网络）以及应用程序开发和控制（软件工具和环境）的基础平台。

软件提供了我们这个时代最重要的产品——信息。它转换个人数据（例如个人财务交易），从而使信息在一定范围内发挥更大的作用；它通过管理商业信息提升竞争力；它为世界范围的信息网络提供通路（例如因特网），并为各类格式的信息提供不同的查询方式。软件还提供了可以威胁个人隐私的载体，并给那些怀有恶意目的的人提供了犯罪的途径。

在最近半个世纪里，计算机软件的作用发生了很大的变化。硬件性能的极大提高、计算机结构的巨大变化、存储容量的大幅度增加以及种类繁多的输入和输出方法都促使基于计算机的系统更加先进和复杂。如果系统开发成功，那么“先进和复杂”可以产生惊人的效果，但是同时复杂性也给系统的开发人员和防护人员带来巨大的挑战。

现在，一个庞大的软件产业已经成为了工业经济中的主导因素。早期的独立程序员也已经被专业的软件开发团队所代替，团队中的不同专业技术人员可分别关注复杂应用系统中的某一部分技术。然而同过去的独立程序员一样，开发现代计算机系统时，软件开发人员依然面临同样的问题：^①

- 为什么软件需要如此长的开发时间？
- 为什么开发成本居高不下？
- 为什么在将软件交付顾客使用之前，我们无法找到所有的错误？
- 为什么维护已有的程序要花费如此多的时间和人工？
- 为什么软件开发和维护的进度仍旧难以度量？

种种问题显示了业界对软件以及软件开发方式的关注，这种关注导致了业界对软件工程实践方法的采纳。

1.1.1 定义软件

今天，绝大多数专业人员和许多普通人认为他们对软件大概了解。真的是这样吗？

来自教科书的关于软件的定义也许是：

软件是：（1）指令的集合（计算机程序），通过执行这些指令可以满足预期的特性、功能和性能需求；（2）数据结构，使得程序可以合理利用信息；（3）软件描述信息，它以硬拷贝和虚拟形式存在，用来描述程序的操作和使用。

关键点 软件既是产品也是交付产品的载体。

引述 软件是播撒梦想和收获噩梦的地方，是一片恶魔和神仙相竞争的抽象而神秘的沼泽，是一个狼人和银弹共存的矛盾世界。

Brad J. Cox

提问 如何定义软件？

^① 在一本优秀的关于软件业务的论文集中，Tom DeMarco[DeM95]提出了相反的看法。他认为：“我们更应该总结使得当今的软件开发费用低廉的成功经验，而不是不停地质问为何软件开发成本高昂。这会有助于我们继续保持软件产业的杰出成就。”

当然，还有更完整的解释。但是一个更加正式的定义可能并不能显著改善其可理解性。为了更好地理解“软件”的含义，有必要将软件和其他人工产品的特点加以区分。软件是逻辑的而非物理的系统元素。因此，软件和硬件具有完全不同的特性：软件不会“磨损”。

图 1-1 描述了硬件的失效率，该失效率是时间的函数。这个名为“浴缸曲线”的关系图显示：硬件在早期具有相对较高的失效率（这种失效通常来自设计或生产缺陷）；在缺陷被逐个纠正之后，失效率随之降低并在一段时间内保持平稳（理想情况下很低）；然而，随着时间推移，因为灰尘、振动、不当使用、温度超限以及其他环境问题所造成的硬件组件损耗累积的效果，使得失效率再次抬高。简而言之，硬件开始磨损了。

而软件是不会被引起硬件磨损的环境问题所影响的。因此，从理论上来说，软件的失效率曲线应该呈现为图 1-2 的“理想曲线”。未知的缺陷将在程序生命周期的前期造成高失效率。然而随着错误的纠正，曲线将如图中所示趋于平缓。“理想曲线”只是软件实际失效模型的粗略简化。曲线的含义很明显——软件不会磨损，但是软件退化的确存在。

建议 若希望降低软件退化，则需要改进软件的设计（第 11 ~ 14 章）。

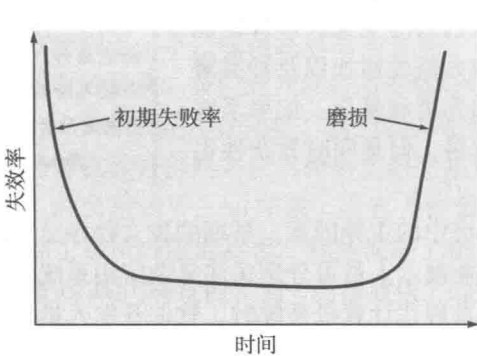


图 1-1 硬件失效曲线图

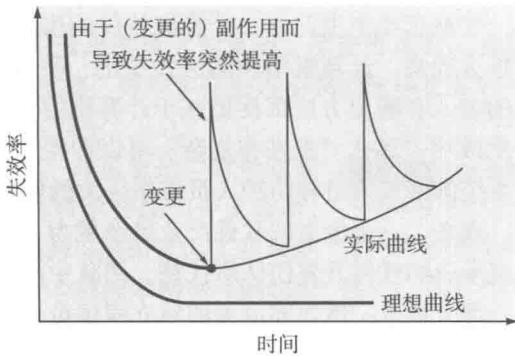


图 1-2 软件失效曲线图

这个似乎矛盾的现象用图 1-2 所示的“实际曲线”可以很好地解释。在完整的生命周期里，[⊖]软件将会面临变更，每次变更都可能引入新的错误，使得失效率像“实际曲线”（图 1-2）那样陡然上升。在曲线回到最初的稳定失效率状态前，新的变更会引起曲线又一次上升。就这样，最小的失效率点沿类似于斜线的形状逐渐上升，可以说，不断的变更是软件退化的根本原因。

关键点 软件工程方法的目的是降低图 1-2 中曲线向上突变的幅度及实际失效曲线的斜率。

磨损的另一方面同样说明了软硬件的不同。磨损的硬件部件可以用备用部件替换，而软件却不存在备用部件。每个软件的缺陷都暗示了设计的缺陷或者在从设计转化到机器可执行代码的过程中产生的错误。因此，软件维护要应对变更请求，比硬件维护更为复杂。

1.1.2 软件应用领域

今天，计算机软件可分为七个大类，软件工程师正面临持续的挑战。

系统软件——整套服务于其他程序的程序。某些系统软件（例如编译器、编辑器、文

⊖ 事实上，在软件开发开始，在第一个版本发布之前的很长时间，许多利益相关者可能提出变更要求。

件管理软件)处理复杂但确定的^①信息结构,另一些系统应用程序(例如操作系统构件、驱动程序、网络软件、远程通信处理器)主要处理的是不确定的数据。

应用软件——解决特定业务需要的独立应用程序。这类应用软件处理商务或技术数据,以协助业务操作或协助做出管理或技术决策。

工程/科学软件——“数值计算”(number crunching)类程序涵盖了广泛的应用领域,从天文学到火山学,从自动压力分析到轨道动力学,从计算机辅助设计到分子生物学,从遗传分析到气象学。

嵌入式软件——嵌入式软件存在于某个产品或者系统中,可实现和控制面向最终用户和系统本身的特性和功能。嵌入式软件可以执行有限的和内部的功能(例如微波炉的按键控制),或者提供重要的功能和控制能力(例如汽车中的燃油控制、仪表板显示、刹车系统等汽车电子功能)。

产品线软件——为多个不同用户的使用提供特定功能。产品线软件关注有限的及内部的市场(例如库存控制产品)或者大众消费品市场。

Web/移动App——以网络为中心,其概念涵盖了宽泛的应用软件产品,包括基于浏览器的App和安装在移动设备上的软件。

人工智能软件——利用非数值算法解决计算和直接分析无法解决的复杂问题。这个领域的应用程序包括机器人、专家系统、模式识别(图像和语音)、人工神经网络、定理证明和博弈等。

全世界成百万的软件工程师在为以上各类软件项目努力地工作着。有时是建立一个新的系统,而有时只是对现有应用程序的纠错、适应性调整和升级。一个年轻的软件工程师所经手项目本身的年限比他自己的年龄还大是常有的事。对于上述讨论的各类软件,上一代的软件工程师都已经留下了遗留系统。我们希望这代工程师留下的遗留系统可以减轻未来工程师的负担。

网络资源 目前最大的共享软件/免费软件库之一: shareware.cnet.com。

引述 对于我来说,电脑是我们能够想出的最重要的工具。它相当于思想的自行车。

Steve Jobs

1.1.3 遗留软件

成千上万的计算机程序都可以归于在前一小节中讨论的7大类应用领域。其中某些是当今最先进的软件——最近才对个人、企业和政府发布。但是另外一些软件则年代较久,甚至过于久远了。

这些旧的系统——通常称为遗留软件(legacy software)——从20世纪60年代起,就成为人们持续关注的焦点。Dayani-Fard和他的同事[Day99]这样描述遗留软件:

遗留软件系统……在几十年前诞生,它们不断被修改以满足商业需要和计算平台的变化。这类系统的繁衍使得大型机构十分头痛,因为它们为维护代价高昂且系统演化风险较高。

Liu和他的同事[Liu98]进一步扩展了这个描述,指出“许多遗留软件系统仍然支持核心的商业功能,是业务必不可少的支撑。”因此,遗留软件具有生命周期长以及业务关键性的特点。

然而不幸的是,遗留软件常常存在另一个特点——质量差^②。遗留软件通常拥有数不清的

① 软件的确定性是指系统的输入、处理和输出的顺序及时间是可以预测的,软件的不确定性是指系统的输入、处理和输出的顺序和时间是无法提前预测的。

② 所谓“质量差”是基于现代软件工程思想的,这个评判标准对遗留系统有些不公平,因为在遗留软件开发的年代里,现代的软件工程一些概念和原则可能还没有被人们完全理解。

问题：遗留系统的设计难以扩展，代码令人费解，文档混乱甚至根本没有，测试用例和结果并未归档，变更的历史管理混乱……然而，这些系统仍然支撑着“核心的业务功能，并且是业务必不可少的支撑”。该如何应对这种情况？

最合理的回答也许就是什么也不做，至少在其不得不进行重大变更之前什么也不做。如果遗留软件可以满足用户的需求并且能可靠运行，那么它就没有失效，不需要修改。然而，随着时间的推移，遗留系统经常会由于下述原因而发生演化：

- 软件需要进行适应性调整，从而可以满足新的计算环境或者技术的需求。
- 软件必须升级以实现新的商业需求。
- 软件必须扩展以使之具有与更多新的系统和数据库的互操作能力。
- 软件架构必须进行改建以使之能适应不断演化的计算环境。

当这些变更发生时，遗留系统需要经过再工程以适应未来的多样性。当代软件工程的目标是“修改在进化论理论上建立的方法论”，即“软件系统不断经历变更，新的软件系统从旧系统中建立起来，并且……新旧所有系统都必须具有互操作性和协作性。” [Day99]。

1.2 软件的变更本质

四大类软件不断演化，在行业中占有主导地位。这四类软件在十几年前还处于初级阶段。

1.2.1 WebApp

万维网（WWW）的早期（大约从1990年到1995年），Web站点仅包含链接在一起的一些超文本文件，这些文件使用文本和有限的图形来表示信息。随着时间的推移，一些开发工具（例如XML、Java）扩展了HTML（超级文本标记语言）的能力，使得Web工程师在向客户提供信息的同时也能提供计算能力。基于Web的系统和应用软件^①（我们将这些总称为WebApp）诞生了。

今天，WebApp已经发展成为成熟的计算工具，这些工具不仅可以为最终用户提供独立的功能，而且已经同公司数据库和业务应用系统集成在一起了。

十年前，WebApp“演化为一种混合体，介于印刷出版和软件开发之间、市场和计算之间内部通信和外部关系之间以及艺术和技术之间。” [Pow98]，但是，如1.1.2节所述，当前WebApp在很多应用类型中提供了丰富的计算能力。

在过去的十多年中，语义Web技术（通常指Web 3.0）已经演化为成熟的企业和消费者应用软件，包括“提供新功能的语义数据库，这些新功能需要Web链接、灵活的数据表示以及外部访问API（应用编程接口）。” [Hen10] 成熟的关系型数据结构导致了全新的WebApp，允许以多种方式访问不同的信息，这在以前是不可能做到的。

提问 如果遇到质量低下的遗留软件该怎么办？

提问 对遗留软件都进行了哪些类型的改变？

建议 所有软件工程师都需认识到变更是不可避免的。不要反对变更。

引述 对于任何类型的稳定性，Web都会变成完全不同的东西。

Louis Monier

① 在本书中，WebApp这个术语包含了很多事物，从一个简单的帮助消费者计算汽车租赁费用的网页，到为商务旅行和度假提供全套旅游服务的大型复杂的Web站点。其中包括完整的Web站点、Web站点的专门功能以及在Internet、Intranet或Extranet上的信息处理应用软件。

1.2.2 移动 App

术语 App 已经演化为在移动平台（例如 iOS、Android 或 Windows Mobile）上专门设计的软件。在大多数情况下，移动 App 包括用户接口，用户接口利用移动平台所提供的独特的交互机制，基于 Web 资源的互操作性提供与 App 相关的大量信息的访问，并具有本地处理能力，以最适合移动平台的方式收集、分析和格式化信息。此外，移动 App 提供了在平台中的持久存储能力。

认识到移动 WebApp 与移动 App 之间的微妙差异是非常重要的。移动 WebApp 允许移动设备通过针对移动平台的优点和弱点专门设计的浏览器获取基于 Web 内容的访问。移动 App 可以直接访问设备的硬件特性（例如加速器或者 GPS 定位），然后提供前面所述的本地处理和存储能力。随着时间的推移，移动浏览器会变得更加成熟，并可获取对设备级硬件和信息的访问，这将使得移动 WebApp 和移动 App 之间的区别变得模糊。

提问 WebApp 和移动 App 之间的差别是什么？

1.2.3 云计算

云计算包括基础设施或“生态系统”，它能使得任何用户在任何地点都可以使用计算设备来共享广泛的计算资源。云计算的总体逻辑结构如图 1-3 所示。

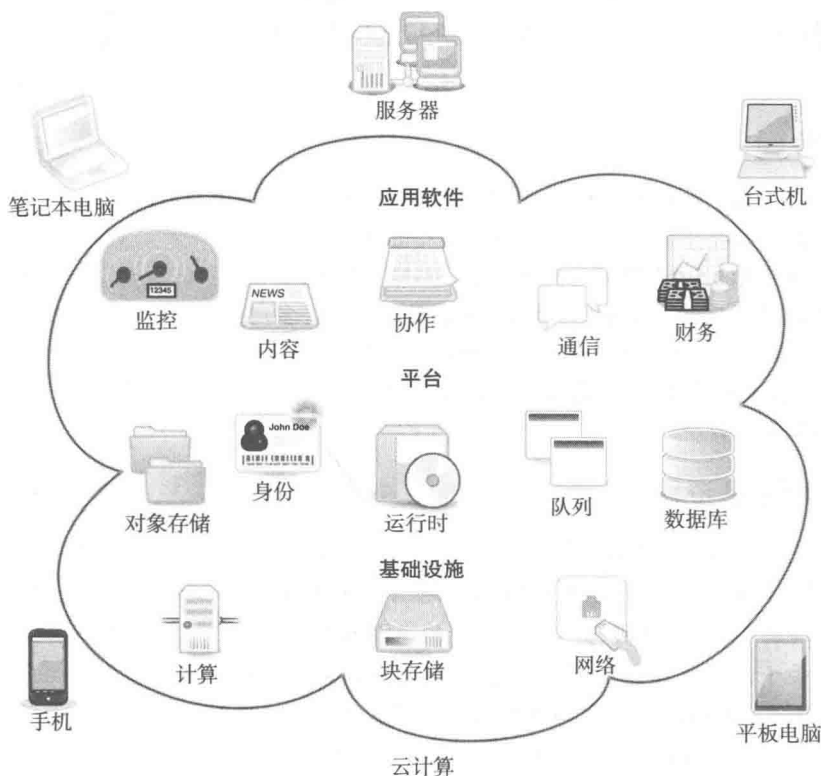


图 1-3 云计算的逻辑结构 [Wik13]

如图所示，计算设备位于云的外部，可以访问云内的各种资源。这些资源包括应用软件、平台和基础设施。最简单的形式是，外部计算设备通过 Web 浏览器或类似的软件访问云。云提供对存储在数据库或其他数据结构中的数据访问。此外，设备可访问可执行的应

用软件，可以用这种应用程序代替计算设备上的 App。

云计算的实现需要开发包含前端和后端服务的体系结构。前端包括客户（用户）设备和应用软件（如浏览器），用于访问后端。后端包括服务器和相关的计算资源、数据存储系统（如数据库）、服务器驻留应用程序和管理服务器。通过建立对云及其驻留资源的一系列访问协议，管理服务器使用中间件对流量进行协调和监控。[Str08]

可以对云体系结构进行分段，以提供不同级别的访问，从公共访问到只对授权用户提供访问的私有云体系结构。

1.2.4 产品线软件

美国卡内基·梅隆大学软件工程研究所（SEI）将软件产品线定义为“一系列软件密集型系统，可以共享一组公共的可管理的特性，这些特性可以满足特定市场或任务的特定需求，并以预定的方法从一组公共的核心资源开发出来。”[SEI13]以某种方式使软件产品相互关联，可见，软件产品线的概念并不是新概念。但是，软件产品线的思想提供了重要的工程影响力，软件产品线都使用相同的底层应用软件和数据体系结构来开发，并使用可在整个产品线中进行复用的一组软件构件来实现。

软件产品线共享一组资源，包括需求（第7章）、体系结构（第12章）、可重用构件（第13章）、测试用例（第17、18章）及其他的软件工作产品。本质上，软件产品线是对很多产品进行开发的结果，在对这些产品进行工程设计时，利用了产品线中所有产品的公共性。

习题与思考题

- 1.1. 举出至少5个例子来说明“意外效应法则”在计算机软件方面的应用。
- 1.2. 举例说明软件对社会的影响（包括正面影响和负面影响）。
- 1.3. 针对1.1节提出的5个问题给出你的答案，并与同学讨论。
- 1.4. 在交付最终用户之前，或者首个版本投入使用之后，许多现代App程序都会有频繁的变更。为防止变更引起软件退化，请提出一些有效的解决措施。
- 1.5. 思考1.1.2节中提到的7个软件分类。请问能否将一个软件工程方法应用于所有的软件分类？并就你的答案加以解释。

扩展阅读与信息资源^①

在数千本关于计算机软件的书中，大多数讨论的是程序设计语言和软件应用系统，很少有涉及软件本身的。Pressman 和 Herron（《Software Shock》，Dorset House，1991）最早讨论了软件和专业开发方法的问题（针对门外汉）。Negroponte 的畅销书（《Being Digital》，Alfred A. Knopf, Inc., 1995）提供了关于计算及其在21世纪的发展和影响的观点。Demarco（《Why does Software Cost So Much?》，Dorset House，1995）就软件和开发过程发表了一系列惊人且见解独到的论文。Ray Kurzweil（《How to Create a Mind》，Viking，2013）讨论了软件如何在不久的将来就会模仿人类思想，并带来人类和机器进化的“奇异性”。

① 在每章小结之后，“扩展阅读与信息资源”一节简单介绍了本章相关资料，便于读者扩展阅读和深入理解本章内容。针对本书，我们已经建立了网站 www.mhhe.com/pressman。网站涉及软件工程的很多主题，逐章列出了相关的软件工程网站资料信息以作为本书的补充，并给出了每一本书在 Amazon.com 的链接。

Keeves (《Catching Digital》, Business Infomedia Online, 2012) 讨论了商业领导者应该如何适应以不断增大的步伐进行演化的软件。Minasi 在著作 (《The Software Conspiracy: Why Software Companies Put Out Faulty Products, How They Can Hurt You, and What You Can Do》, McGraw-Hill, 2000) 中认为, 现在由于软件缺陷引起的“现代灾难”将被消除并提出了解决的方法。Eubanks (《Digital Dead End: Fighting for Social Justice in the Information Age》, MIT Press, 2011) 和 Compaine (《Digital Divide: Facing a Crisis or Creating a Myth》, MIT Press, 2001) 的书认为, 在 21 世纪的第一个十年里, 信息 (如 Web 资源) 富有者和信息贫困者之间的数字鸿沟将越来越小。Kuniavsky (《Smart Things: Ubiquitous Computing User Experience Design》, Morgan Kaufman, 2010)、Greenfield (《Everyware: The Dawning Age of Ubiquitous Computing》, New Riders Publishing, 2006) 和 Loke (《Context-Aware Pervasive Systems: Architectures for a New Breed of Applications》, Auerbach, 2006) 的著作介绍了“开放世界”软件的概念, 并指出在无线网络环境中软件必须能够进行适应性调整, 以满足实时涌现的需求。

网上有很多讨论软件本质的信息资源, 与软件过程相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

软件工程

要点浏览

概念: 软件工程包括过程、一系列方法(实践)和大量工具,专业人员借由这些来构建高质量的计算机软件。

人员: 软件工程师应用软件工程过程。

重要性: 软件工程之所以重要是因为它使得我们可以高效、高质量地构建复杂系统。它使杂乱无章的工作变得有序,但也允许计算机软件的创建者调整其工作方式,以更好地适应要求。

步骤: 开发计算机软件就像开发任何成功的产品一样,需采用灵活、可适应的软

件开发过程,完成可满足使用者要求的高质量 的软件产品。这就是软件工程化方法。

工作产品: 从软件工程师的角度来看,工作产品 是计算机软件,包括程序、内容(数据)和其他工作产品;而从用户的角度来看,工作产品是可以改善生活和工作质量的最终信息。

质量保证措施: 阅读本书的后面部分,选择切合你所构建的软件特点的思想,并在实际工作中加以应用。

要构建能够适应 21 世纪挑战的软件产品,我们必须认识到以下几个简单的事实:

- 软件实际上已经深入到我们生活的各个方面,其结果是,对软件应用所提供的特性和功能感兴趣的人显著增多。[⊖]因此,在确定软件方案之前,需要共同努力来理解问题。
- 年复一年,个人、企业以及政府的信息技术需求日臻复杂。过去一个人可以构建的计算机程序,现在需要由一个庞大的团队共同实现。曾经在可预测、独立的计算环境中实现的复杂软件,现在要将其嵌入任何产品中,从消费性电子产品到医疗器械再到武器系统。因此,设计已经成为关键活动。
- 个人、企业和政府在进行日常运作管理以及战略战术决策时越来越依赖于软件。软件失效会给个人和企业带来诸多不便,甚至是灾难性的失败。因此,软件应该具有高质量。
- 随着特定应用系统感知价值的提升,其用户群和软件寿命也会增加。随着用户群和使用时间的增加,其适应性和增强型性需求也会同时增加。因此,软件需具备可维护性。

由这些简单事实可以得出一个结论:各种形式、各个应用领域的软件

关键概念

框架活动
通用原则
原则
问题解决
SafeHome
软件工程
定义
层次
实践
软件神话
软件过程
普适性活动

关键点 在建立解决方案之前应理解问题。

关键点 质量和可维护性都是良好设计的产物。

⊖ 在本书的后续部分,将这些人统称为“利益相关者”。

都需要工程化。这也是本书的主题——软件工程。

2.1 定义软件工程学科

对于软件工程,美国电气与电子工程师学会(IEEE) [IEE93a] 给出了如下定义:

软件工程是:(1)将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护,即将工程化方法应用于软件;(2)对(1)中所述方法的研究。

然而,对于某个软件开发队伍来说可能是“系统化的、规范的、可量化的”方法,对于另外一个团队却可能是负担。因此,我们需要规范,也需要可适应性和灵活性。

软件工程是一种层次化的技术,如图2-1所示。任何工程方法(包括软件工程)必须构建在质量承诺的基础之上。全面质量管理、六西格玛和类似的理念^①促进了持续不断的过程改进文化,正是这种文化最终引导人们开发出更有效的软件工程方法。支持软件工程的根基在于质量关注点(quality focus)。

软件工程的基础是过程(process)层。软件过程将各个技术层次结合在一起,使得合理、及时地开发计算机软件成为可能。过程定义了一个框架,构建该框架是有效实施软件工程技术必不可少的。软件过程构成了软件项目管理控制的基础,建立了工作环境以便于应用技术方法、提交工作产品(模型、文档、数据、报告、表格等)、建立里程碑、保证质量及正确的管理变更。

软件工程方法(method)为构建软件提供技术上的解决方法(如何做)。方法覆盖面很广,包括沟通、需求分析、设计建模、程序构造、测试和技术支持。软件工程方法依赖于一组基本原则,这些原则涵盖了软件工程所有技术领域,包括建模活动和其他描述性技术等。

软件工程工具(tool)为过程和方法提供自动化或半自动化的支持。这些工具可以集成起来,使得一个工具产生的信息可被另外一个工具使用,这样就建立了软件开发的支撑系统,称为计算机辅助软件工程(computer-aided software engineering)。

2.2 软件过程

软件过程是工作产品构建时所执行的一系列活动、动作和任务的集合。活动(activity)主要实现宽泛的目标(如与利益相关者进行沟通),与应用领域、项目大小、结果复杂性或者实施软件工程的重要程度没有直接关系。动作(action,如体系结构设计)包含了主要工作产品(如体系结构设计模型)生产过程中的一系列任务。任务(task)关注小而明确的目标,能够产生实际产品(如构建一个单元测试)。

提问 如何定义软件工程?

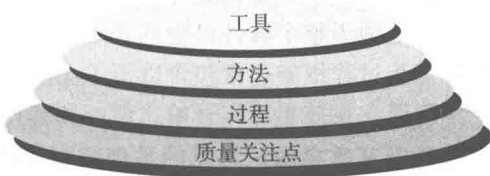


图 2-1 软件工程层次图

关键点 软件工程包括过程、管理和构建软件的方法和工具。

网络资源
《Cross Talk》杂志提供了关于过程、方法和工具的许多实践信息,网站地址是 www.stsc.hill.af.mil。

提问 软件过程的元素是什么?

引述 过程定义了活动的时间、人员、工作内容和达到预期目标的途径。

Ivar Jacobson,
Grandy Booch,
and James
Rumbaugh

① 质量管理和相关方法的内容在本书第三部分进行讨论。

在软件工程领域，过程不是对如何构建计算机软件的严格的规定，而是一种具有可适应性的调整方法，以便于工作人员（软件团队）可以挑选适合的工作动作和任务集合。其目标通常是及时、高质量地交付软件，以满足软件项目资助方和最终用户的需求。

2.2.1 过程框架

过程框架（process framework）定义了若干个框架活动（framework activity），为实现完整的软件工程过程建立了基础。这些活动可广泛应用于所有软件开发项目，无论项目的规模和复杂性如何。此外，过程框架还包含一些适用于整个软件过程的普适性活动（umbrella activity）。一个通用的软件工程过程框架通常包含以下 5 个活动。

沟通。在技术工作开始之前，和客户（及其他利益相关者^①）的沟通与协作是极其重要的，其目的是理解利益相关者的项目目标，并收集需求以定义软件特性和功能。

策划。如果有地图，任何复杂的旅程都可以变得简单。软件项目好比是一个复杂的旅程，策划活动就是创建一个“地图”，以指导团队的项目旅程，这个地图称为软件项目计划，它定义和描述了软件工程师工作，包括需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度计划。

建模。无论你是庭园设计师、桥梁建造者、航空工程师、木匠还是建筑师，你每天的工作都离不开模型。你会画一张草图来辅助理解整个项目大的构想——体系结构、不同的构件如何结合，以及其他一些特性。如果需要，可以把草图不断细化，以便更好地理解问题并找到解决方案。软件工程师也是如此，需要利用模型来更好地理解软件需求，并完成符合这些需求的软件设计。

构建。必须要对所做的设计进行构建，包括编码（手写的或者自动生成的）和测试，后者用于发现编码中的错误。

部署。软件（全部或者部分增量）交付给用户，用户对其进行评测并给出反馈意见。

上述五个通用框架活动既适用于简单小程序的开发，也可用于 WebApp 的建造以及基于计算机的大型复杂系统工程。不同的应用案例中，软件过程的细节可能差别很大，但是框架活动都是一致的。

对许多软件项目来说，随着项目的开展，框架活动可以迭代应用。也就是说，在项目的多次迭代过程中，沟通、策划、建模、构建、部署等活动不断重复。每次项目迭代都会产生一个软件增量（software increment），每个软件增量实现了部分的软件特性和功能。随着每一次增量的产生，软件将逐渐完善。

2.2.2 普适性活动

软件工程过程框架活动由很多普适性活动来补充实现。通常，这些普适性活动贯穿软件项目始终，以帮助软件团队管理和控制项目进度、质量、变更和风险。典型的普适性活动包括如下活动。

提问 五个最基本的过程框架活动是什么？

引述 爱因斯坦认为必然存在着一个对自然界的简单解释，因为上帝既不专制也不喜怒无常。然而软件工程师却无法抱侥幸心理，多数情况下，他必须面对毫无规律的复杂性。

Fred Brooks

^① 利益相关者（stakeholder）就是可在项目成功中分享利益的人，包括业务经理、最终用户、软件工程师、支持人员等。Rob Thomsett 曾开玩笑说：“stakeholder 就是掌握巨额投资（stake）的人……如不照看好你的 stakeholder，就会失去投资。”

软件项目跟踪和控制——项目组根据计划来评估项目进度，并且采取必要的措施保证项目按进度计划进行。

风险管理——对可能影响项目成果或者产品质量的风险进行评估。

软件质量保证——确定和执行保证软件质量的活动。

技术评审——评估软件工程产品，尽量在错误传播到下一个活动之前发现并清除错误。

测量——定义和收集过程、项目以及产品的度量，以帮助团队在发布软件时满足利益相关者的要求。同时，测量还可与其他框架活动和普适性活动配合使用。

软件配置管理——在整个软件过程中管理变更所带来的影响。

可复用管理——定义工作产品复用的标准（包括软件构件），并且建立构件复用机制。

工作产品的准备和生产——包括生成产品（如建模、文档、日志、表格和列表等）所必需的活动。

上述每一种普适性活动都将在本书后续部分详细讨论。

关键点 普适性

活动贯穿整个软件过程，主要关注项目管理、跟踪和控制。

关键点 对软件过程的适应性调整是项目成功的关键。

2.2.3 过程的适应性调整

在本节前面部分曾提到，软件工程过程并不是教条的法则，也不要求软件团队机械地执行；而应该是灵活可适应的（根据软件所需解决的问题、项目特点、开发团队和组织文化等进行适应性调整）。因此，不同项目所采用的项目过程可能有很大不同。这些不同主要体现在以下几个方面：

- 活动、动作和任务的总体流程以及相互依赖关系。
- 在每一个框架活动中，动作和任务细化的程度。
- 工作产品的定义和要求的程度。
- 质量保证活动应用的方式。
- 项目跟踪和控制活动应用的方式。
- 过程描述的详细程度和严谨程度。
- 客户和利益相关者对项目的参与程度。
- 软件团队所赋予的自主权。
- 队伍组织和角色的明确程度。

本书第1部分将详细介绍软件过程。

引述 我认为食谱只是指导方法，一个聪明的厨师每次都会变化出不同的特色。

Madame Benoit

2.3 软件工程实践

在2.2节中，曾介绍过一种由一组活动组成的通用软件过程模型，建立了软件工程实践的框架。通用的框架活动——**沟通、策划、建模、构建和部署**——和普适性活动构成了软件工程工作的体系结构的轮廓。但是软件工程的实践如何融入该框架呢？在以下几节里，读者将会对应用于这些框架活动的基本概念和原则有一个基本了解。^①

网络资源 软件

工程实践方面各种深刻的想法都可以在以下网址获得：www.literateprogramming.com。

① 在本书的后面对于特定软件工程方法和普适性活动的讨论中，你应该重读本章中的相关章节。

2.3.1 实践的精髓

在现代计算机发明之前，有一本经典著作《How to Solve it》，在书中，George Polya[Pol45]列出了解决问题的精髓，这也正是软件工程实践的精髓：

1. 理解问题（沟通和分析）。
2. 策划解决方案（建模和软件设计）。
3. 实施计划（代码生成）。
4. 检查结果的正确性（测试和质量保证）。

在软件工程中，这些常识性步骤引发了一系列基本问题（与 [Pol45] 相对应）：

理解问题。虽然不愿承认，但生活中的问题很多都源于我们的傲慢。

我们只听了几秒钟就断言，好，我懂了，让我们开始解决这个问题吧。不幸的是，理解一个问题不总是那么容易，需要花一点时间回答几个简单问题：

- 谁将从问题的解决中获益？也就是说，谁是利益相关者？
- 有哪些是未知的？哪些数据、功能和特性是解决问题所必需的？
- 问题可以划分吗？是否可以描述为更小、更容易理解的问题？
- 问题可以图形化描述吗？可以建立分析模型吗？

策划解决方案。现在你理解了要解决的问题（或者你这样认为），并迫不及待地开始编码。在编码之前，稍稍慢下来做一点点设计：

- 以前曾经见过类似问题吗？在潜在的解决方案中，是否可以识别一些模式？是否已经有软件实现了所需要的数据、功能和特性？
- 类似问题是否解决过？如果是，解决方案所包含元素是否可以复用？
- 可以定义子问题吗？如果可以，子问题是否已有解决方案？
- 能用一种可以很快实现的方式来描述解决方案吗？能构建出设计模型吗？

实施计划。前面所创建的设计勾画了所要构建的系统的路线图。可能存在没有想到的路径，也可能在实施过程中会发现更好的解决路径，但是这个计划可以保证在实施过程中不至迷失方向。需要考虑的问题是：

- 解决方案和计划一致吗？源码是否可追溯到设计模型？
- 解决方案的每个组成部分是否可以证明正确？设计和代码是否经过评审？或者采用更好的方式，算法是否经过正确性证明？

检查结果。你不能保证解决方案是最完美的，但是可以保证设计足够的测试来发现尽可能多的错误。为此，需回答：

- 能否测试解决方案的每个部分？是否实现了合理的测试策略？
- 解决方案是否产生了与所要求的数据、功能和特性一致的结果？是否按照项目利益相关者的需求进行了确认？

不足为奇，上述方法大多是常识。但实际上，有充足的理由可以证明，在软件工程中采用常识将让你永远不会迷失方向。

建议 你可能会觉得 Polya 的方法只是简单的常识，的确如此。但是令人惊奇的是，在软件世界里，很多常识常常不为人知。

建议 理解问题最重要的是倾听。

引述 在任何问题的解决方案中都会有所发现。

George Polya

2.3.2 通用原则

原则这个词在字典里的定义是“某种思想体系所需要的重要的根本规则或者假设”。在

本书中，我们将讨论一些不同抽象层次上的原则。一些原则关注软件工程的整体，另一些原则考虑特定的、通用的框架活动（比如**沟通**），还有一些关注软件工程的动作（比如架构设计）或者技术任务（比如编制用例场景）。无论关注哪个层次，原则都可以帮助我们建立一种思维方式，进行扎实的软件工程实践。因此，原则非常重要。

David Hooker[Hoo96]提出了7个关注软件工程整体实践的原则，这里复述如下。[⊖]

第1原则：存在价值

一个软件系统因能为用户提供价值而具有存在价值，所有的决策都应该基于这个思想。在确定系统需求之前，在关注系统功能之前，在决定硬件平台或者开发过程之前，问问你自己：这确实能为系统增加真正的价值吗？如果答案是不，那就坚决不做。所有的其他原则都以这条原则为基础。

第2原则：保持简洁

软件设计并不是一种随意的过程，在软件设计中需要考虑很多因素。所有的设计都应该尽可能简洁，但不是过于简化。这有助于构建更易于理解和易于维护的系统。这并不是说有些特性（甚至是内部特性）应该以“简洁”为借口而取消。的确，优雅的设计通常也是简洁的设计，但简洁也不意味着“快速和粗糙”。事实上，它经常是经过大量思考和多次工作迭代才达到的，这样做的回报是所得到的软件更易于维护且错误更少。

第3原则：保持愿景

清晰的愿景是软件项目成功的基础。没有愿景，项目将会由于它有“两种或者更多种思想”而永远不能结束；如果缺乏概念的一致性，系统就好像是由许多不协调的设计补丁、错误的集成方式强行拼凑在一起……如果不能保持软件系统体系架构的愿景，就会削弱甚至彻底破坏设计良好的系统。授权体系架构师，使其能够持有愿景，并保证系统实现始终与愿景保持一致，这对项目开发的成功至关重要。

第4原则：关注使用者

有产业实力的软件系统不是在真空中开发和使用的。通常软件系统必定是由开发者以外的人员使用、维护和编制文档等，因此必须要让别人理解你的系统。在需求说明、设计和实现过程中，牢记要让别人理解你所做的事情。对于任何一个软件产品，其工作产品都可能有很多用户。需求说明时应时刻想到用户，设计中始终想到实现，编码时想着那些要维护和扩展系统的人。一些人可能不得不调试你所编写的代码，这使得他们成了你所编写代码的使用者，尽可能地使他们的工作简单化会大大提升系统的价值。

第5原则：面向未来

生命周期持久的系统具有更高的价值。在现今的计算环境中，需求规格说明随时会改变，硬件平台几个月后就会淘汰，软件生命周期都是以月而不是以年来衡量的。然而，真正具有“产业实力”的软件系统必须持久耐用。为了成功地做到这一点，系统必须能适应各种变化，能成功做到这一点的系统都是那些一开始就以这种路线来设计的系统。永远不要把自己的设计局限于一隅，经常问问“如果出现……应该怎样应对”，构建可以解决通用问题的

建议 在开始一个软件项目之前，应首先确保软件具有商业目标并且让用户体会到它的价值。

提问 简洁比所有巧妙的措词更加美妙。

Alexander Pope
(1688-1744)

关键点 如果软件有价值，那么其价值在其生命周期中将发生变更。因此，软件必须构建成可维护的。

⊖ 这里的引用得到了作者的授权 [Hoo96].Hooker 定义这些原则的模式请参见：<http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment>。

系统,为各种可能的方案做好准备,^①这很可能会提高整个系统的可复用性。

第6原则:提前计划复用

复用既省时又省力^②。软件系统开发过程中,高水平的复用是很难实现的一个目标。曾有人宣称代码和设计复用是面向对象技术带来的主要好处,然而,这种投入的回报不会自动实现。为达到面向对象(或是传统)程序设计技术所提供的复用性,需要有前瞻性的设计和计划。系统开发过程中的各个层面上都有多种技术可实现复用……提前做好复用计划将降低开发费用,并增加可复用构件以及构件化系统的价值。

第7原则:认真思考

这最后一条规则可能是最容易被忽略的。在行动之前清晰定位、完整思考通常能产生更好的结果。仔细思考可以提高做好事情的可能性,而且也能获得更多的知识,明确如何把事情做好。如果仔细思考过后还是把事情做错了,那么,这就变成了很有价值的经验。思考就是学习和了解本来一无所知的事情,使其成为研究答案的起点。把明确的思想应用在系统中,就产生了价值。使用前六条原则需要认真思考,这将带来巨大的潜在回报。

如果每位工程师、每个开发团队都能遵从 Hooker 这七条简单的原则,那么,开发复杂计算机软件系统时所遇到的许多困难都可以迎刃而解。

2.4 软件开发神话

软件开发神话,即关于软件及其开发过程的一些被人盲目相信的说法,这可以追溯到计算技术发展的初期。神话具有一些特点,让人觉得不可捉摸。例如,神话看起来是事实的合理描述(有时确实包含真实的成分),它们符合直觉,并且经常被那些知根知底的有经验的从业人员拿来宣传。

今天,大多数有见地的软件工程师已经意识到软件神话的本质——它实际上误导了管理者和从业人员对软件开发的態度,从而引发了严重的问题。然而,由于习惯和态度的根深蒂固,软件神话遗风犹在。

管理神话。像所有领域的经理一样,承担软件职责的项目经理肩负着维持预算、保证进度和提高质量的压力。就像溺水人抓住稻草一样,软件经理经常依赖软件神话中的信条,只要它能够减轻以上的压力(即使是暂时性的)。

神话:我们已经有了——一本写满软件开发标准和规程的宝典,难道不能提供我们所需要了解的所有信息吗?

事实:这本宝典也许的确已经存在,但它是否在实际中采用了?从业人员是否知道这本书的存在呢?它是否反映了软件工程的现状?是否全面?是否可以适应不同的应用环境?是否在缩短交付时间的同时还关注产品质量的保证?在很多情况下,问题的答案是否定的。

神话:如果我们未能按时完成计划,我们可以通过增加程序员人数而赶上进度(即所谓的“蒙古游牧”概念)。

事实:软件开发并不是像机器制造那样的机械过程。Brooks 曾说过 [Bro95]:“在软件工

网络资源 软件项目经理网有助于人们澄清这些神话: www.spmn.com。

① 把这个建议发挥到极致会非常危险,设计通用方案会带来性能损失,并降低特定的解决方案的效率。

② 尽管对于准备在未来的项目中复用软件的人而言,这种说法是正确的,但对于设计和实现可复用构件的人来说,复用的代价会很昂贵。研究表明,设计和开发可复用构件比直接开发目标软件要增加 25% ~ 200% 的成本,在有些情况下,这些费用差别很难核实。

程中，为赶进度而增加人手只能使进度更加延误。”初看，这种说法似乎与直觉不符。然而，当新人加入到一个软件项目后，原有的开发人员必须要牺牲本来的开发时间对后来者进行培训，因此减少了本应用于高效开发的时间。只有在有计划且有序进行的情况下，增加人员对项目进度才有意义。

神话：如果决定将软件外包给第三方公司，就可以放手不管，完全交给第三方公司开发。

事实：如果开发团队不了解如何在内部管理和控制软件项目，那么将无一例外地在外包项目中遇到困难。

客户神话。软件产品的客户可能是隔壁的某个人、楼下的一个技术团队、市场/销售部门或者签订软件合同的某个外部公司。多数情况下，客户之所以相信所谓的软件神话，是因为项目经理和从业人员没有及时纠正他们的错误信息。软件神话导致客户错误的期望，最终导致对开发者的不满。

神话：有了对项目目标的大概了解，便足以开始编写程序，可以在之后的项目开发过程中逐步充实细节。

事实：虽然通常很难得到综合全面且稳定不变的需求描述，但是对项目目标模糊不清的描述将为项目实施带来灾难。要得到清晰的需求描述（经常是逐步变得清晰的），只能通过客户和开发人员之间的持续有效的沟通。

神话：虽然软件需求不断变更，但是因为软件是弹性的，因此可以很容易地适应变更。

事实：软件需求的确在随时变更，但随变更引入的时机不同，变更所造成的影响也不同。如果需求变更提出得较早（比如在设计或者代码开发之前），则对费用的影响较小^①；但是，随着时间的推移，变更的代价也迅速增加——因为资源已经被分配，设计框架已经建立，而变更可能会引起的剧变，需要添加额外的资源或者修改主要设计。

从业者神话。在60多年的编程文化的滋养下，软件开发人员依然深信着各种神话。在软件业发展早期，编程被视为一种艺术。旧有的方式和态度根深蒂固。

神话：当我们完成程序并将其交付使用之后，我们的任务就完成了。

事实：曾经有人说过，对于编程来说，开始得越早，耗费的时间就越长。业界的一些数据显示，60%～80%的工作耗费在软件首次交付顾客使用之后。

神话：直到程序开始运行，才能评估其质量。

事实：最有效的软件质量保证机制之一——技术评审，可以从项目启动就开始实行。软件评审作为“质量过滤器”，已经证明其可以比软件测试更为有效地发现多种类型的软件缺陷。

神话：对于一个成功的软件项目，可执行程序是唯一可交付的工作成果。

事实：软件配置包括很多内容，可执行程序只是其中之一。各种工作产品（如模型、文档、计划）是成功实施软件工程的基础，更重要的是，为软件技术支持提供了指导。

建议 在项目开始之前，尽可能努力了解工作内容。也许难以明确所有细节，但你了解得越多，所面临的风险就越低。

建议 每当你认为没有时间采用软件工程方法时，就再问问自己：“是否有时间重做整个软件？”

① 许多软件工程师采纳了“敏捷”（agile）开发方法，通过增量的方式逐步纳入变更，以便控制变更的影响范围和成本。本书第5章讨论敏捷方法。

神话: 软件工程将导致我们产生大量无用文档, 并因此降低工作效率。

事实: 软件工程并非以创建文档为目的, 而是为了保证软件产品的开发质量。好的质量可以减少返工, 从而加快交付时间。

目前, 大多数软件专业人员已经认识到软件神话的谬误。对于软件开发真实情况的正确理解是系统阐述如何使用软件工程方法解决实际问题的第一步。

2.5 这一切是如何开始的

每个软件工程项目都来自业务需求——对现有应用程序缺陷的纠正, 改变遗留系统以适应新的业务环境, 扩展现有应用程序功能和特性, 或者开发某种新的产品、服务或系统。

在软件项目的初期, 业务需求通常是在简短的谈话过程中非正式地表达出来的。以下这段简短谈话就是一个典型的例子。

SafeHome[⊖] 如何开始一个软件项目

[场景] CPI 公司的会议室里。CPI 是一个虚构的为家庭和贸易应用生产消费产品的公司。

[人物] Mal Golden, 产品开发部高级经理; Lisa Perez, 营销经理; Lee Warren, 工程经理; Joe Camalleri, 业务发展部执行副总裁。

[对话]

Joe: Lee, 我听说你们那帮家伙正在开发一个产品——通用的无线盒?

Lee: 哦, 是的, 那是一个很棒的产品, 只有火柴盒大小。我们可以把它放在各种传感器上, 比如数码相机, 总之任何东西里。采用 802.11n 无线网络协议, 可以通过无线连接获得它的输出。我们认为它可以带来全新的一代产品。

Joe: Mal, 你觉得怎么样呢?

Mal: 我当然同意。事实上, 随着这一年来销售业绩的趋缓, 我们需要一些新的产品。Lisa 和我已经做了一些市场调查, 我们都认为该系列产品具有很大的市场潜力。

Joe: 多大, 底线是多少?

Mal (避免直接承诺): Lisa, 和他谈谈我们的想法。

Lisa: 这是新一代的家庭管理产品, 我们称之为“SafeHome”。产品采用新型无线接口, 给家庭和小型商务从业人士提供一个由电脑控制的系统——住宅安全、监视, 仪表和设备控制。例如, 你可以在回家的路上关闭家里的空调, 或者如此这类的应用。

Lee (插话): Joe, 工程部已经作了相关的技术可行性研究。它可行且制造成本不高。大多数硬件可以在市场购买产品, 不过软件方面是个问题, 但也不是我们不能做的。

Joe: 有意思! 我想知道底线。

Mal: 在美国, 70% 的家庭拥有电脑。如果我们定价合适, 这将成为一个十分成功的产品。到目前为止, 只有我们拥有这一无线控制盒技术。我们将在这方面保持两年的领先地位。收入吗, 在第二年大约可达到 3000 万到 4000 万。

Joe (微笑): 我很感兴趣, 让我们继续讨论一下。

⊖ SafeHome 项目将作为一个案例贯穿本书, 以便说明项目组在开发软件产品过程中的内部工作方式。公司、项目和人员都是虚构的, 但场景和问题是真实的。

除了一带而过地谈到软件,这段谈话中几乎没有提及软件开发项目。然而,软件将是 SafeHome 产品线成败的关键。只有 SafeHome 软件成功,该产品才能成功。只有嵌入其中的软件产品满足顾客的需求(尽管还未明确说明),产品才能被市场所接受。我们将在后面的几章中继续讨论 SafeHome 中软件工程的话题。

习题与思考题

- 2.1 图 2-1 中,将软件工程的三个层次放在了“质量关注点”这层之上。这意味着在整个开发组织内采用质量管理活动,如“全面质量管理”。仔细研究并列出全面质量管理活动中关键原则的大纲。
- 2.2 软件工程对构建 WebApp 是否适用?如果适用,需要如何改进以适应 WebApp 的独特特点?
- 2.3 随着软件的普及,由于程序错误所带来的公众风险已经成为一个愈加重要的问题。设想一个真实场景:由于软件错误而引起“世界末日”般的重大危害(危害社会经济或是人类生命财产安全)。
- 2.4 用自己的话描述过程框架。当我们谈到框架活动适用于所有的项目时,是否意味着对于不同规模和复杂度的项目可应用相同的工作任务?请解释。
- 2.5 普适性活动存在于整个软件过程中,你认为它们均匀分布于软件过程中,还是集中在某个或者某些框架活动中?
- 2.6 在 2.4 节所列举的神话中,增加两种软件神话,同时指出与其相对应的真实情况。

扩展阅读与信息资源

软件工程及软件过程的当前发展状况可以参阅一些期刊,如《IEEE Software》《IEEE Computer》《CrossTalk》和《IEEE Transactions on Software Engineering》。《Application Development Trends》和《Cutter IT Journal》等行业期刊通常包含一些关于软件工程的文章。每年,IEEE 和 ACM 资助的研讨会论文集《Proceeding of the International Conference on Software Engineering》都是对当年学术成果的总结,并且在《ACM Transactions on Software Engineering and Methodology》《ACM Software Engineering Notes》和《Annals of Software Engineering》等期刊上有进一步深入讨论。当然,在互联网上有很多关于软件工程和软件过程的网页。

近年出版了许多关于软件过程和软件工程的书籍,有些是关于整个过程的概要介绍,有些则深入讨论过程中一些重要专题。下面是一些畅销书(除本书之外):

《SWEBOK: Guide to the Software Engineering Body of Knowledge》^①, IEEE, 2013, 见 <http://www.computer.org/portal/web/swebok>。

Andersson, E. 等,《Software Engineering for Internet Applications》, MIT Press, 2006。

Braude, E. 和 M. Bernstein,《Software Engineering: Modern Approaches》, 2nd ed., Wiley, 2010。

Christensen, M. 和 R. Thayer,《A Project Manager's Guide to Software Engineering Best Practices》, IEEE-CS Press (Wiley), 2002。

Glass, R.,《Fact and Fallacies of Software Engineering》, Addison-Wesley, 2002。

Hussain, S.,《Software Engineering》, I K International Publishing House, 2013。

Jacobson, I.,《Object-Oriented Software Engineering: A Use Case Driven Approach》, 2nd ed., Addison-Wesley, 2008。

Jalote, P.,《An Integrated Approach to Software Engineering》, 3rd ed., Springer, 2010。

^① 可从 <http://www.computer.org/portal/web/swebok/htmlformat> 免费下载。

Pfleeger, S.,《Software Engineering: Theory and Practice》, 4th ed., Prentice Hall, 2009。

Schach, S.,《Object-Oriented and Classical Software Engineering》, 8th ed., McGraw-Hill, 2010)。

Sommerville, I.,《Software Engineering》, 9th ed., Addison-Wesley, 2010。

Stober, T., 和 U. Hansmann,《Agile Software Development: Best Practices for Large Development Projects》, Springer, 2009。

Tsui, F., 和 O.karam,《Essentials of Software Engineering》, 2nd ed., Jones&Bartlett Publishers, 2009。

Nygaard(《Release it!: Design and Deploy Production-Ready Software》, Pragmatic Bookshelf, 2007)、Richardson 和 Gwaltney (《Ship it! A Practical Guide to Successful Software Projects》, Pragmatic Bookshelf, 2005) 以及 Humble 和 Farley (《Continues Delivery: Reliable Software Releases through Build, Test, and Deployment Automation》, Addison-Wesley, 2010) 的书给出了大量有用的指导原则, 可用于部署活动。

在过去的几十年里, IEEE、ISO 以及附属其下的标准化组织发布了大量软件工程标准。Moore (《The Road Map to Software Engineering: A Standards-Based Guide》, IEEE Computer Society Press[Wiley], 2006) 对相关标准进行了调查并指出了这些标准应如何应用到实际工程中。

网上有很多有关软件工程和软件过程相关问题的信息资源, 与软件过程相关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

软件过程

本部分将介绍软件过程，它提供了软件工程实践的框架。在本部分的各章中将涉及以下问题：

- 什么是软件过程？
- 软件过程中，有哪些通用的框架活动？
- 如何建立过程模型？什么是过程模式？
- 什么是惯用过程模型？有哪些优缺点？
- 为什么现代软件工程关注敏捷问题？
- 什么是敏捷软件开发？它与传统的过程模型有什么区别？

在解决了上述问题之后，就可以对软件工程实践的应用背景有更清楚的认识。

软件过程结构

要点浏览

概念：在开发产品或构建系统时，遵循一系列可预测的步骤（即路线图）是非常重要的，它有助于及时交付高质量的产品。软件开发中所遵循的路线图就称为“软件过程”。

人员：软件工程师及其管理人员根据需要调整开发过程，并遵循该过程。除此之外，软件的需求方也需要参与过程的定义、建立和测试。

重要性：软件过程提高了软件工程活动的稳定性、可控性和有组织性，如果不进行控制，软件活动将变得混乱。但是，现代软件工程方法必须是“灵活”的，也就是要求软件工程活动、控制以及工作

产品适合于项目团队和将要开发的产品。

步骤：具体来讲，采用的过程依赖于构造软件的特点。飞机航空系统的软件与网站的建设可能需要采用两种截然不同的软件过程。

工作产品：从软件工程师的角度来看，工作产品体现为在执行过程所定义的任务和活动的过程中，所产生的程序、文档和数据。

质量保证措施：有大量的软件过程评估机制，开发机构可以评估其软件过程的“成熟度”。然而，表征软件过程有效性的最好指标还是所构建产品的质量、及时性和寿命。

Howard Baetjer Jr.[Bae98] 曾著书从经济学家的角度分析软件和软件工程，该书引人入胜，对软件过程评述如下：

软件同其他资产一样，是知识的具体体现，而知识最初都是以分散的、不明确的、隐蔽的且不完整的形式广泛存在的，因此，软件开发是一个社会学习的过程。软件过程是一个对话的过程，在对话中，获取需要转化为软件的知识，并在软件中实现这些知识。软件过程提供了用户与设计人员之间、用户与不断演化的工具之间以及设计人员与不断演化的工具（技术）之间的互动。软件开发是一个迭代的过程，在其中演化的工具本身就作为沟通的媒介，任何新一轮对话都可以从参与的人员中获得更有用的知识。

构建计算机软件确实是一个迭代的社会学习的过程，其输出——即 Baetjer 所称的“软件资产”——是知识的载体，这些知识在过程执行中进行收集、提炼和组织。

但从技术的角度如何确切地定义软件过程呢？本书将软件过程定义为一个为创建高质量软件所需要完成的活动、动作和任务的框架。过程与软件工程同义吗？答案是“是，也不是”。软件过程定义了软件工程化中采用的方法，但软件工程还包含该过程中应用的技术——技术方法和自动化工具。

更重要的是，软件工程是由有创造力、有知识的人完成的，他们根据产品构建的需要和

关键概念

通用过程模型
过程评估
过程流
过程改进
过程模式
任务集

市场需求来选取成熟的软件过程。

3.1 通用过程模型

在第2章中，过程定义为在工作产品构建过程中所需完成的工作活动、动作和任务的集合。这些活动、动作、任务中的每一个都隶属于某一框架或者模型，框架或模型定义了它们与过程之间或者相互之间的关系。

软件过程示意图如图3-1所示。由图可以看出，每个框架活动由一系列软件工程动作构成；每个软件工程动作由任务集来定义，这个任务集明确了将要完成的工作任务、将要产生的工作产品、所需要的质量保证点，以及用于表明过程状态的里程碑。

关键点 在软件过程中，技术工作的层次包括活动，活动由动作构成，动作由任务构成。

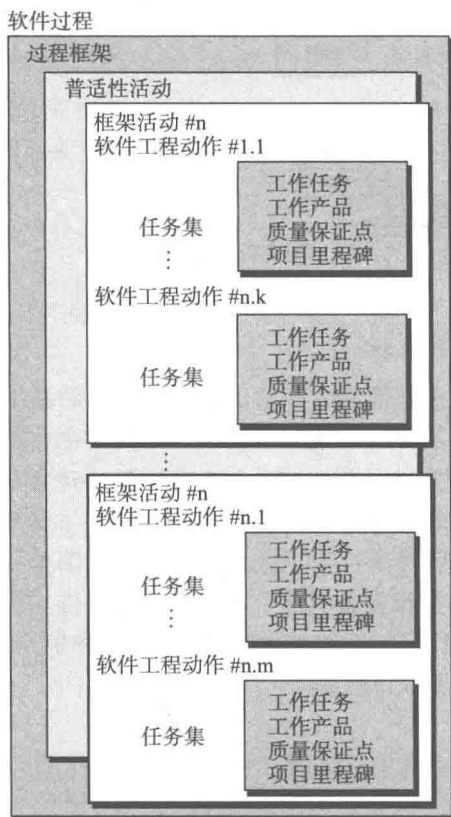


图 3-1 软件过程框架

正如在第2章中讨论的，软件工程的通用过程框架定义了五种框架活动——沟通、策划、建模、构建以及部署。此外，一系列普适性活动——项目跟踪控制、风险管理、质量保证、配置管理、技术评审以及其他活动——贯穿软件过程始终。

你也许注意到了，软件过程的一个很重要的方面还没有讨论，即过程流（process flow）。过程流描述了在执行顺序和执行时间上如何组织框架中的活动、动作和任务，如图3-2所示。

提问 什么是过程流？

线性过程流（linear process flow）从沟通到部署顺序执行五个框架活动（参见图3-2a）。迭代过程流（iterative process flow）在执行下一个活动前重复执行之前的一个或多个活动

(参见图 3-2b)。演化过程流 (evolutionary process flow) 采用循环的方式执行各个活动，每次循环都能产生更为完善的软件版本 (参见图 3-2c)。并行过程流 (parallel process flow) (参见图 3-2d) 将一个或多个活动与其他活动并行执行 (例如，软件一个方面的建模可以同软件另一个方面的构建活动并行执行)。

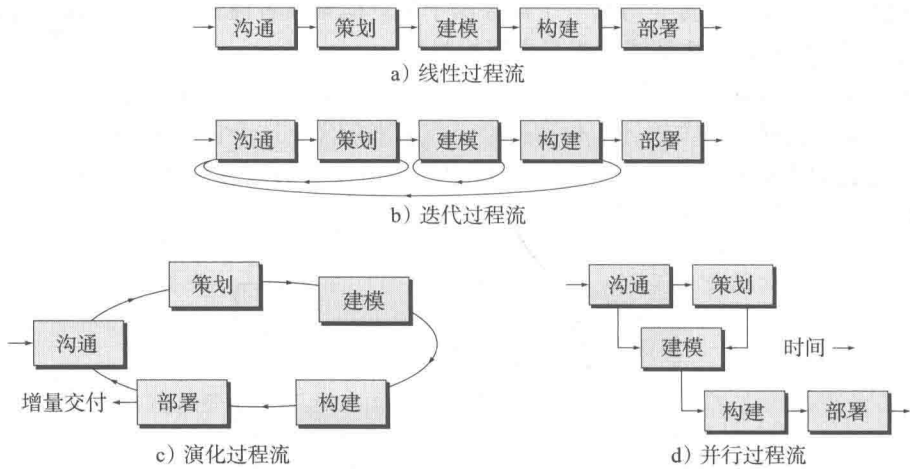


图 3-2 过程流

3.2 定义框架活动

尽管第 2 章描述了 5 种框架活动，并给出了每种活动的基本定义，但是软件团队要在软件过程中具体执行这些活动中的任何一个，还需要更多信息。因此，我们面临一个关键问题：针对给定的问题、开发人员和利益相关者，哪些动作适合于框架活动？

对于由个人负责的小型软件项目（可能远程），其需求简单明确，沟通活动也许仅仅是与合适的利益相关者的一个电话或一封邮件。因此，主要的动作是电话交流，这个动作所包括的主要工作任务（任务集）有：

- 1. 通过电话与利益相关者取得联系。
- 2. 讨论需求并做记录。
- 3. 将笔记整理成一份简单的书面需求。
- 4. 通过 E-mail 请利益相关者审阅并认可。

如果项目有多个利益相关者，则要复杂得多，每个参与人员都有着不同的需求（有时这些需求甚至是相互冲突的），沟通活动可能会包含 6 个不同的动作（具体参见第 7 章）：起始、需求获取、需求细化、协商、规格说明和确认。每个软件工程师动作都可能有很多工作任务和一些不同的工作产品。

3.3 明确任务集

我们再来看图 3-1，每一个软件工程师动作（如需求获取以及与沟通活动相关的动作）都由若干个任务集（task set）构成，而每一个任务集都由软件工程师工作任务、相关工作产品、质量保证点和项目里程碑组成。需要选择最能满足项目需要并适合开发团队特点的任

引述 如果过程正确，就会得到应得的结果。

Takashi Osada

提问 框架活动如何随着项目性质的变化而变化？

关键点 不同的项目需要不同的任务集。软件团队根据问题和项目的特点选择任务集。

务集。这就意味着软件工程动作可以根据软件项目的特定需要和项目团队的特点做适当的调整。

信息栏 任务集

任务集定义了为达到一个软件工程动作的目标所需要完成的工作。例如,需求获取(通常称为“需求收集”)就是发生在沟通活动中的一个重要的软件工程动作。需求获取的目的是理解利益相关者对将构建的软件的需求。

对于一个小型、相对简单的项目而言,需求获取的任务集可能包括:

1. 制定项目的利益相关者列表。
2. 邀请所有的利益相关者参加一个非正式会议。
3. 征询每个人对于软件特性和功能的需求。
4. 讨论需求,并确定最终的需求列表。
5. 划定需求优先级。
6. 标出不确定域。

对于大型、复杂的软件工程项目而言,可能需要如下不同的任务集:

1. 制定项目的利益相关者列表。
2. 和利益相关者的每个成员分别单独讨论,获取所有的要求。

3. 基于利益相关者的输入,建立初步的功能和特性列表。
4. 安排一系列促进需求获取的会议。
5. 组织会议。
6. 在每次会议上建立非正式的用户场景。
7. 根据利益相关者的反馈,进一步细化用户场景。
8. 建立一个修正的利益相关者需求列表。
9. 使用质量功能部署技术,划分需求优先级。
10. 将需求打包以便于软件可以实施增量交付。
11. 标注系统的约束和限制。
12. 讨论系统验证方法。

上面两种任务集都可以完成需求获取,但是无论从深度还是形式化的程度上来说,二者都有很大区别。软件团队采取适当的任务集以达到每个动作的目的,并且保持软件质量和开发的敏捷性。

3.4 过程模式

每个软件团队在软件过程里都会遇到很多问题。针对这些问题,如果软件团队能够得到已有的经过验证的解决方案,将有助于他们快速地分析和解决问题。过程模式(process pattern)^①描述了软件工程中遇到的过程相关的问题,明确了问题环境并给出了针对该问题的一种或几种可证明的解决方案。通俗地讲,过程模式提供了一个模板[Amb98]——一种在软件过程的背景下统一描述问题解决的方法。通过模式组合,软件团队可以解决问题并定义最符合项目需求的开发过程。

我们可以在不同抽象层次上定义模式^②。在某些情况下,模式可以描述一个与完整过程模型(例如原型开发)相关的问题(及其解决方案);在其他的情况下,模式可以描述一个与

提问 什么是过程模式?

① 关于模式的详细讨论参见第10章。

② 模式的概念广泛应用于软件工程的活动中。第10、12、14章将分别讨论分析模式、设计模式和测试模式。本书第四部分将讨论项目管理活动中的模式和反模式。

框架活动(如策划)或者框架活动中的一动作(如项目估算)相关的问题(及其解决方案)。

Ambler[Amb98]提出了下面的过程模式的描述模板:

模式名称。模式名称应能清楚地表述该模式在软件过程中的含义(例如技术评审)。

驱动力。模式的使用环境及主要问题,这些问题会显现在软件过程中并可能影响解决方案。

类型。定义模式类型。Ambler[Amb98]提出了三种类型:

1. **步骤模式(stage pattern)**——定义了与过程的框架活动相关的问题。由于框架活动包括很多动作和工作任务,因此步骤模式包括与步骤(框架活动)有关的许多任务模式(见以下描述)。例如,建立沟通可能作为一个步骤模式,该步骤模式可能包括需求收集等任务模式。
2. **任务模式(task pattern)**——定义了与软件工程动作或是工作任务相关、关系软件工程实践成败的问题(例如,需求收集是一个任务模式)。
3. **阶段模式(phase pattern)**——定义在过程中发生的框架活动序列,即使这些活动流本质上是迭代的。例如,螺旋模型和原型开发^①就可能是两种阶段模式。

启动条件。它描述的是模式应用的前提条件。在应用模式之前需要明确:(1)在此之前,整个开发组织或是开发团队内已经有哪些活动?(2)过程的进入状态是什么?(3)已经有哪些软件工程信息或是项目信息?

例如,策划模式(阶段模式)需要的前提条件有:(1)客户和软件工程师已经建立了合作的交流机制;(2)已经成功完成一些客户沟通模式中特定的任务模式;(3)项目范围、基本业务需求和项目限制条件已经确定。

问题。描述模式将要解决的具体问题。

解决方案。描述如何成功实现模式。这部分主要讨论随着模式的启动,过程的初始状态(模式应用之前就已经存在)是如何发生改变的。解决方案也描述了随着模式的成功执行,模式启动之前所获得的软件工程信息和项目信息是如何变换的。

结果。描述模式成功执行之后的结果。模式完成时需要明确:(1)必须完成哪些开发组织或是开发团队相关的活动?(2)过程的结束状态是什么?(3)产生了哪些软件工程信息或是项目信息?

相关模式。以层次化或其他图的方式列举与该模式相关的其他过程模式。例如步骤模式沟通包括了一组任务模式:项目团队组织、合作指导原则定义、范围分解、需求收集、约束描述以及场景模式的创建等。

已知应用和实例。说明该模式可应用的具体实例。例如,沟通在每一个软件项目的开始都是必需的,建议在整个软件项目过程中采用,并规定在部署活动中必须进行。

过程模式提供了一种有效的机制,用以解决任何与软件过程相关的问题。模式使得软件工程组织能够从高层抽象开始(阶段模式)建立层次化的过程描述。高层抽象描述又进一步细化为一系列步骤模式以描述框架活动,然后每一个步骤模式又进一步逐层细化为更详细的任务模式。过程模

引述 模式的重复与软件模块的重复完全不同。事实上,不同的模块是独特的,而模式却是相同的。

Christopher
Alexander

关键点 模式模板提供了描述模式的一般性方法。

引述 我们认为软件开发人员遗漏了一个事实:多数机构并不清楚他们在做什么,他们以为清楚了,但实际上不清楚。

Tom DeMarco

网络资源 可在下面的网站查看过程模式的全面资源: www.amblysoft.com/processPatternsPage.html。

① 第4章将讨论这些阶段模式。

式一旦建立起来,就可以进行复用以定义各种过程变体,即软件开发团队可以将模式作为过程模型的构建模块,定制特定的过程模型。

信息栏 过程模式实例

当利益相关者对工作成果有大致的想法,但对具体的软件需求还不确定时,下述简化的过程模式描述了可采用的方法。

模式名称。需求不清。

目的。该模式描述了一种构建模型(或是原型系统)的方法,使得利益相关者可以反复评估,以便识别和确定软件需求。

类型。阶段模式。

启动条件。在模式启动之前必须满足以下四个条件:(1)确定利益相关者;(2)已经建立起利益相关者和软件开发团队之间的沟通方式;(3)利益相关者确定了需要解决的主要问题;(4)对项目范围、基本业务需求和项目约束条件有了初步了解。

问题。需求模糊或者不存在,但都清楚地认识到项目存在问题,且该问题需要

通过软件解决。利益相关者不确定他们想要什么,即他们无法详细描述软件需求。

解决方案。描述了原型开发过程,详见 4.1.3 节。

结果。开发了软件原型,识别了基本的需求(例如交互模式、计算特性、处理功能等),并获得了利益相关者的认可。随后,可能有两种结果:(1)原型系统可以通过一系列的增量开发,演化成为软件产品;(2)原型系统被抛弃,采用其他过程模式建立了产品软件。

相关模式。以下模式与该模式相关:客户沟通,迭代设计,迭代开发,客户评价,需求抽取。

已知应用和实例。当需求不确定时,推荐原型开发方法。

习题与思考题

- 3.1 在本章的介绍中,Baetjer 说过:“软件过程提供了用户与设计人员之间、用户与开发工具之间以及设计人员与开发工具之间的互动。”对以下四个方面各设计五个问题:(1)设计人员应该问用户的;(2)用户应该问设计人员的;(3)用户对将要构建的软件的自问;(4)设计人员对于软件产品和建造该产品采取的软件过程的自问。
- 3.2 讨论 3.1 节所描述的不同过程流之间的区别。你是否能够确定适用于所描述的每种通用流的问题类型?
- 3.3 为沟通活动设计一系列动作,选定一个动作作为其设计一个任务集。
- 3.4 在沟通过程中,遇到两位对软件如何做有着不同想法的利益相关者是很常见的问题。也就是说,你得到了相互冲突的需求。设计一种过程模式(可以是步骤模式),利用 3.4 节中针对此类问题的模板,给出一种行之有效的解决方法。

扩展阅读与信息资源

大多数软件工程课本都会详细介绍过程模型。Sommerville(《Software Engineering》, 9th ed., Addison-Wesley, 2010)、Schach(《Object-Oriented and Classical Software Engineering》, 8th ed., McGraw-Hill, 2010)以及Pfleeger和Atlee(《Software Engineering: Theory and Practice》, 4th ed., Prentice Hall, 2009)的书中介绍了这些传统的模型,并讨论了它们的优点和缺点。Munch和他的同事(《Software Process Definition and Management》, Springer, 2012)介绍了过程和产品的软件和系

统工程观点。Glass (《 Facts and Fallacies of Software Engineering 》, Prentice-Hall, 2002) 提出了一种保证软件工程过程的不加修饰且实用的观点。Brooks (《 The Mythical Man-Month 》, 2d ed., Addison-Wesley, 1995) 在他的书中虽然没有直接讲过程, 但是他用一生的项目学识讲了和过程相关的每一个方面。

Firesmith 和 Henderson-Sellers (《 The OPEN Process Framework: An introduction 》, Addison-Wesley, 2001) 为创建“灵活但有序的软件过程”提出了一个通用的模板, 并讨论了过程属性和目的。Madachy (《 Software Process Dynamics 》, Wiley-IEEE, 2008) 讨论了一种对软件过程中的相关技术和社会因素进行分析的建模技术。Sharpe 和 McDermott (《 Workflow Modeling: Tools for Process Improvement and Application Development 》, 2nd ed., Artech House, 2008) 介绍了为软件和商业过程建模的工具。

网上有大量关于软件工程和软件过程的信息, 和软件过程有关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

过程模型

要点浏览

概念: 过程模型为软件工程师工作提供了特定的路线图, 该路线图规定了所有活动的流程、动作、任务、迭代的程度、工作产品及要完成的工作应如何组织。

人员: 软件工程师及其管理人员根据他们的要求采用一种过程模型, 并遵循该过程模型。此外, 软件的需求方也需要参与过程的定义、构建和测试。

重要性: 软件过程提高了软件工程活动的稳定性、可控性和有组织性, 如果没有过程约束, 软件活动将失控并变得混乱。但是, 现代软件工程方法必须是“灵活”的, 也就是要求软件工程活动、控制以

及工作产品适合于项目团队和将要开发的产品。

步骤: 过程模型为软件人员提供了开展软件工程师工作需要遵循的步骤。

工作产品: 从软件工程师的角度来看, 工作产品是对过程所定义的活动和任务的格式化描述。

质量保证措施: 有大量的软件过程评估机制, 开发机构可以评估其软件过程的“成熟度”。然而, 表征软件过程有效性的最好指标还是所构建产品的质量、及时性和寿命。

最早提出过程模型是为了改变软件开发的混乱状况, 使软件开发更加有序。历史证明, 这些模型为软件工程师工作提出了大量有用的结构, 并为软件团队提供了有效的路线图。尽管如此, 软件工程师工作及其产品仍然停留在“混乱的边缘”。

在一篇探讨软件世界中有序和混乱之间奇怪关系的论文中, Nogueira和他的同事指出 [Nog00]:

混乱的边缘可定义为“有序和混乱之间的一种自然状态, 结构化和反常之间的重大妥协” [Kau95]。混乱的边缘可以被视为一种不稳定和部分结构化的状态……它的不稳定是因为它不停地受到混乱或者完全有序的影响。

我们通常认为有序是自然的完美状态。这可能是个误区……研究证实, 打破平衡的活动会产生创造力、自我组织的过程和更高的回报 [Roo96]。完全的有序意味着缺乏可变性, 而可变性在某些不可预测的环境下往往是一种优势。变更通常发生在某些结构中, 这些结构使得变更可以被有效组织, 但还不是死板得使得变更无法发生。另一方面, 太多的混乱会使协调和一致成为不可能。缺少结构并不意味着无序。

上面这段话的哲学思想对于软件工程有着重要的意义。本章所描述的每个过程模型都试图在找出混乱世界中的秩序和适应不断发生的变化这两种要求之间寻求平衡。

关键概念

面向方面的软件开发
基于构件的开发
并发模型
演化过程模型
形式化方法模型
增量过程模型
过程建模工具
原型开发
螺旋模型
统一过程
V 模型
瀑布模型

4.1 惯用过程模型

惯用过程模型^①力求达到软件开发的结构和秩序，其活动和任务都是按照过程的特定指引顺序进行的。但是，对于富于变化的软件世界，这一模型是否合适呢？如果我们抛弃传统过程模型（以及模型所规定的秩序），以一些不够结构化的模型取而代之，是否会使软件工作无法达到协调和一致？

这些问题无法简单回答，但是软件工程师有很大的选择余地。在接下来的章节中，我们将探讨以秩序和一致性作为主要问题的传统过程方法。我们称为“传统”是因为，它规定了一套过程元素——框架活动、软件工程动作、任务、工作产品、质量保证以及每个项目的变更控制机制。每个过程模型还定义了过程流（也称为工作流）——也就是过程元素相互之间关联的方式。

所有的软件过程模型都支持第2章和第3章中描述的通用框架活动，但是每一个模型都对框架活动有不同的侧重，并且定义了不同的过程流以不同的方式执行每一个框架活动（以及软件工程动作和任务）。

4.1.1 瀑布模型

有时候，当从沟通到部署都采用合理的线性 workflows 方式的时候，可以清楚地理解问题的需求。这种情况通常发生在需要对一个已经存在的系统进行明确定义的适应性调整或是增强的时候（比如政府修改了法规，导致财务软件必须进行相应修改）；也可能发生在很少数新的开发工作上，但是需求必须是准确定义和相对稳定的。

瀑布模型（waterfall model）又称为经典生命周期（classic life cycle），它提出了一个系统的、顺序的软件开发方法^②，从用户需求规格说明开始，通过策划、建模、构建和部署的过程，最终提供完整的软件支持（图4-1）。

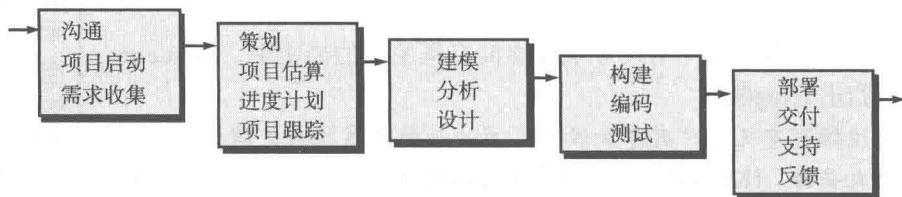


图 4-1 瀑布模型

瀑布模型的一个变体称为 V 模型（V-model）。如图 4-2 所示，V 模型 [Buc99] 描述了质量保证动作同沟通、建模相关动作以及早期构建相关的动作之间的关系。随着软件团队工作沿着 V 模型左侧步骤向下推进，基本问题需求逐步细化，形成了对问题及解决方案的详尽且技术性的描述。一

关键点 过程模型的作用是减少开发新软件产品时出现的混乱。

网络资源 包括最重要的惯用过程模型的“过程模拟比赛”获奖模型可见于 <http://www.ics.uci.edu/~emilyyo/SimSE/downloads.html>。

关键点 惯用过程模型定义了一组规定的过程元素和一个可预测的过程工作流。

关键点 V 模型阐明了验证和确认动作如何与早期工程动作相互关联。

① 惯用过程模型有时称为“传统”过程模型。

② 尽管对 Winston Royce [Roy70] 提出的最早的瀑布模型进行了改进，加入了“反馈”循环，但绝大多数软件组织在应用该过程模型时都将其视为严格的线性模型。

一旦编码结束，团队沿着 V 模型右侧的步骤向上推进工作，其本质上是执行了一系列测试（质量保证动作），这些测试验证了团队沿着 V 模型左侧步骤向下推进过程中所生成的每个模型^①。实际上，经典生命周期模型和 V 模型没有本质区别，V 模型提供了一种将验证和确认动作应用于早期软件工程工作中的直观方法。

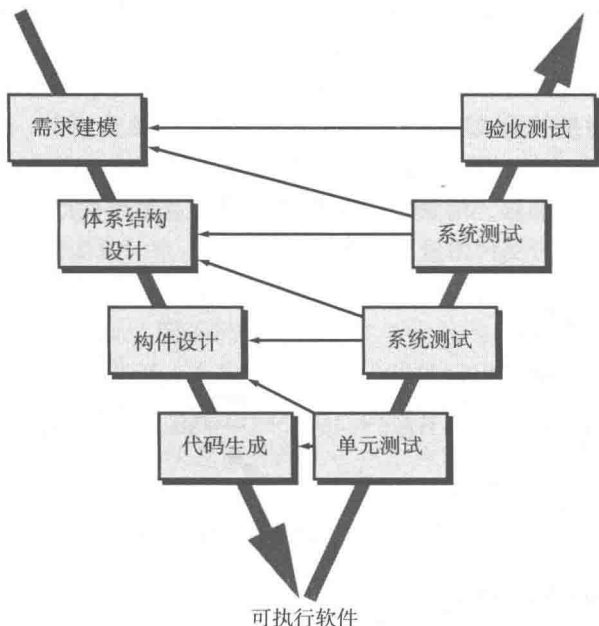


图 4-2 V 模型

瀑布模型是软件工程最早的范例。尽管如此，在过去的 40 多年中，对这一过程模型的批评使它最热情的支持者都开始质疑其有效性 [Han95]。在运用瀑布模型的过程中，人们遇到的问题包括：

1. 实际的项目很少遵守瀑布模型提出的顺序。虽然线性模型可以加入迭代，但是它是用间接的方式实现的，结果是，随着项目组工作的推进，变更可能造成混乱。
2. 客户通常难以清楚地描述所有的需求。而瀑布模型却要求客户明确需求，这就很难适应在许多项目开始阶段必然存在的不确定性。
3. 客户必须要有耐心，因为只有在项目接近尾声的时候，他们才能得到可执行的程序。对于系统中存在的重大缺陷，如果在可执行程序评审之前没有发现，将可能造成惨重损失。

在分析一个实际项目时，Bradac[Bra94] 发现，经典生命周期模型的线性特性在某些项目中会导致“阻塞状态”，由于任务之间的依赖性，开发团队的一些成员要等待另一些成员工作完成。事实上，花在等待上的时间可能超过花在生产性工作上的时间。在线性过程的开始和结束，这种阻塞状态更容易发生。

目前，软件工作快速进展，经常面临永不停止的变更流，特性、功能

提问 为什么瀑布模型有时候会失效？

引述 大多数情况下，软件工作遵从骑自行车第一定律：不论你去哪，你都会顶风骑上坡路。

作者不详

① 质量保证动作的详细讨论参见本书第三部分。

和信息内容都会变更,瀑布模型往往并不适合这类工作。尽管如此,在需求已确定的情况下,且工作采用线性的方式完成的时候,瀑布模型是一个很有用的过程模型。

4.1.2 增量过程模型

在许多情况下,初始的软件需求有明确的定义,但是整个开发过程却不宜单纯运用线性模型。同时,可能迫切需要为用户迅速提供一套功能有限的软件产品,然后在后续版本中再进行细化和扩展功能。在这种条件下,需要选用一种以增量的形式生产软件产品的过程模型。

增量模型综合了第3章讨论的线性过程流和并行过程流的特征。如图4-3所示,随着时间的推移,增量模型在每个阶段都运用线性序列。每个线性序列生产出软件的可交付增量 [McD93]。

关键点 增量模型交付一系列称为增量的版本,随着每个版本的交付,逐步为用户提供更多的功能。

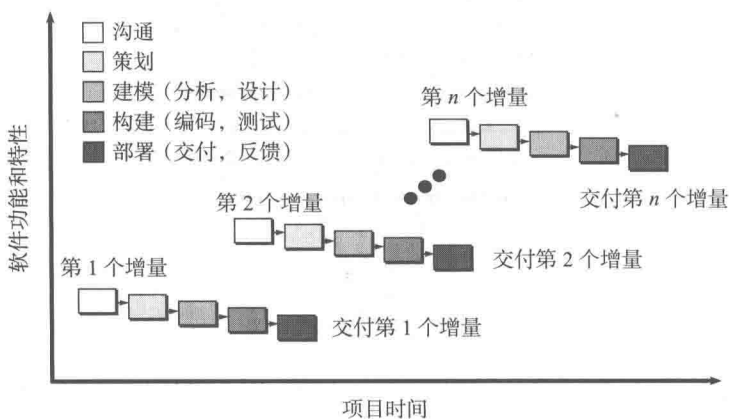


图 4-3 增量模型

例如,采用增量模型开发文字处理软件,在第一个增量中提供基本的文件管理、编辑和文档生成功能;在第二个增量中提供更为复杂的编辑和文档生成功能;在第三个增量中提供拼写和语法检查功能;在第四个增量中提供高级页面排版功能。需要注意的是,任何增量的过程流都可能使用下一节中讨论的原型范型。

运用增量模型的时候,第一个增量往往是核心产品 (core product)。也就是满足了基本的需求,但是许多附加的特性(一些是已知的,一些是未知的)没有提供,客户使用该核心产品并进行仔细的评估,然后根据评估结果制定下一个增量计划。这份计划应说明需要对核心产品进行的修改,以便更好地满足客户的要求,也应说明需要增加的特性和功能。每一个增量的交付都会重复这一过程,直到最终产品的产生。

建议 如果你的客户要求你在一个不可能完成的时间提交产品,那么向他建议届时只提交一个或几个增量,此后再提交软件的其他增量。

4.1.3 演化过程模型

软件类似于其他复杂的系统,会随着时间的推移而演化。在开发过程中,商业和产品需求经常发生变化,这将直接导致最终产品难以实现;严格的交付时间使得开发团队不可能圆满完成综合性的软件产品,但是必须交付功能有限的版本以应对竞争或商业压力;虽然能很好地理解核心产品和系统需求,但是产品或系统扩展的细节问题却没有定义。在上述情况和

关键点 演化过程模型中,每个迭代产生软件的一个更完整的版本。

类似情况下，软件开发人员需要一种专门应对不断演变的软件产品的过程模型。

演化模型是迭代的过程模型，这种模型使得软件开发人员能够逐步开发出更完整的软件版本。接下来，将介绍两种常用的演化过程模型。

原型开发。很多时候，客户定义了软件的一些基本任务，但是没有详细定义功能和特性需求。另一种情况下，开发人员可能对算法的效率、操作系统的适用性和人机交互的形式等情况并没有把握。在这些情况和类似情况下，采用**原型开发范型**（prototyping paradigm）是最好的解决办法。

虽然原型可以作为一个独立的过程模型，但是更多的时候是作为一种技术，可以在本章讨论的任何一种过程模型中应用。不论人们以什么方式运用它，当需求很模糊的时候，原型开发模型都能帮助软件开发人员和利益相关者更好地理解究竟需要做什么。

原型开发范型（图 4-4）开始于沟通。软件开发人员和其他利益相关者进行会晤，定义软件的整体目标，明确已知的需求，并大致勾画出以后再进一步定义的东西。然后迅速策划一个原型开发迭代并进行建模（以“快速设计”的方式）。快速设计要集中在那些最终用户能够看到的方面（比如人机接口布局或者输出显示格式）。快速设计产生了一个原型。对原型进行部署，然后由利益相关者进行评估。根据利益相关者的反馈信息，进一步精炼软件的需求。在原型系统不断调整以满足各种利益相关者需求的过程中，采用迭代技术，同时也使开发者逐步清楚用户的需求。

理想状况下，原型系统提供了定义软件需求的一种机制。当需要构建可执行的原型系统时，软件开发人员可以利用已有的程序片段或应用工具快速产生可执行的程序。

如果我们的原型达到了上述目的，那么接下来它有什么用呢？Brooks[Bro95]给出了答案：

在大多数项目中，构建的第一个系统很少是好用的，可能太慢了、太大了、太难用了，或者同时具备上述三点。除了重新开始，没有更好的选择。采用更巧妙的方法来构建一个重新设计的版本，解决上述问题。

原型可以作为第一个系统，也就是 Brooks 推荐我们扔掉的系统。但这可能是一种理想的方式。尽管许多原型系统是临时系统并且会被废弃，但其他一些原型系统将会演化为实际系统。

利益相关者和软件工程师确实都喜欢原型开发范型。客户对实际的系统有了直观的认识，开发者也迅速建立了一些东西。但是，原型开发也存在一些问题，原因如下。

1. 利益相关者看到了软件的工作版本，却未察觉到整个软件是随意搭成的，也未察觉到为了尽快完成软件，开发者没有考虑整体软件质

引述 有时候你很想扔掉一款产品，但是通常情况下，你唯一的选择是考虑如何将它卖给客户。

Frederick P.
Brooks

建议 如果你的客户有一个合理的要求，但是对细节没有思路，那么不妨先开发一个原型。

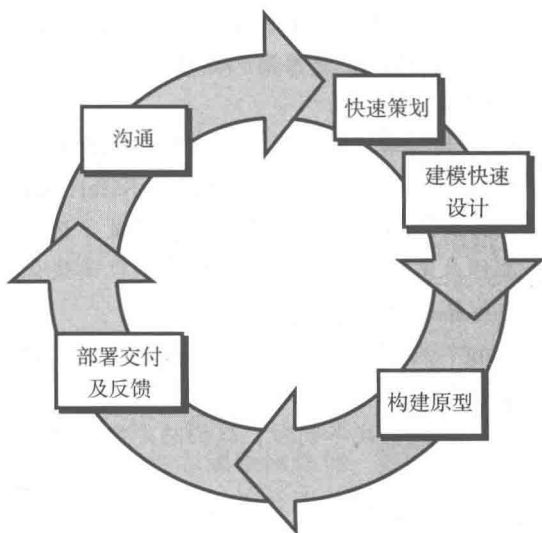


图 4-4 原型开发范型

建议 面对把一个粗糙的原型系统变为工作产品的压力时，如果采取抵制态度，那么结果往往是产品质量受到损害。

量和长期的可维护性。当开发者告诉客户整个系统需要重建以提高软件质量的时候,利益相关者会不愿意,并且要求对软件稍加修改使其变为一个可运行的产品。在绝大多数的情况下,软件开发管理层会做出妥协。

2. 作为一名软件工程师,为了使一个原型快速运行起来,往往在实现过程中采用折衷的手段。他们经常会使用不合适的操作系统或程序设计语言,仅仅因为当时可用或他们对此较为熟悉。他们也经常会采用一种低效的算法,仅为了证明系统的能力。时间长了,软件开发人员可能会适应这些选择,而忽略了这些选择其实并不合适的理由,结果使并不完美的选择成了系统的组成部分。

尽管问题会发生,但原型开发对于软件工程来说仍是一个有效的范型。关键是要在游戏开始的时候制定规则,也就是说,所有利益相关者必须承认原型是为定义需求服务的。然后丢弃原型(至少是部分丢弃),实际的软件系统是以质量第一为目标而开发的。

SafeHome 选择过程模型 第1部分

[场景] CPI 公司软件工程部会议室。该(虚构的)公司专注于开发家用和商用的消费产品。

[人物] Lee Warren, 工程经理; Doug Miller, 软件工程经理; Jamie Lazar、Vinod Raman 和 Ed Robbins, 软件团队成员。

[对话]

Lee: 我简单说一下。正如我们现在所看到的,我已经花了很多时间讨论 SafeHome 产品的产品线。毫无疑问,我们做了很多工作来定义这个东西,我想请各位谈谈你们打算如何做这个产品的软件部分。

Doug: 看起来,我们过去在软件开发方面相当混乱。

Ed: Doug, 我不明白,我们总是能成功开发出产品来。

Doug: 你说的是事实,不过我们的开发工作并不是一帆风顺,并且我们这次要做的项目看起来比以前做的任何项目都要庞大和复杂。

Jamie: 没有你说的那么严重,但是我同意你的看法。我们过去混乱的项目开发方法这次行不通了,特别是这次我们的时间很紧。

Doug (笑): 我希望我们的开发方法更专业一些。我上星期参加了一个培训班,学到了很多关于软件工程的知识。我们现在需要一个过程。

Jamie (皱眉): 我的工作编程,不是文书。

Doug: 在你反对我之前,请先尝试一下。我想说的是……(Doug 开始讲述第3章讲述的过程框架和本章到目前为止讲到的惯用过程模型)

Doug: 所以,似乎线性模型并不适合我们……它假设我们此刻明确了所有的需求,而事实上并不是这样。

Vinod: 同意你的观点。线性模型太 IT 化了……也许适合于开发一套库存管理系统或者什么,但是不适合我们的 SafeHome 产品。

Doug: 对。

Ed: 原型开发方法听起来不错,正适合我们现在的处境。

Vinod: 有个问题,我担心它不够结构化。

Doug: 别担心。我们还有许多其他选择。我希望在座的各位选出最适合我们小组和我们这个项目的开发范型。

螺旋模型。最早由 Barry Boehm[Boe88] 提出,螺旋模型是一种演进式软件过程模型。它结合了原型的迭代性质和瀑布模型的可控性和系统性特点。它具有快速开发越来越完善的软件版本的潜力。Boehm[Boe01a] 如下这样描述螺旋模型:

螺旋模型是一种风险驱动型的过程模型生成器,对于软件集中的系统,它可以指导多个利益相关者的协同工作。它有两个显著的特点。一是采用循环的方式逐步加深系统定义和实现的深度,同时降低风险。二是确定一系列里程碑作为支撑点,确保利益相关者认可是可行的且可令各方满意的系统解决方案。

螺旋模型将软件开发为一系列演进版本。在早期的迭代中,软件可能是一个理论模型或是原型。在后来的迭代中,会产生一系列逐渐完整的系统版本。

螺旋模型被分割成一系列由软件工程团队定义的框架活动。为了讲解方便,我们使用前文讨论的通用框架活动^①。如图 4-5 所示,每个框架活动代表螺旋上的一个片段。随着演进过程开始,从圆心开始顺时针方向,软件团队执行螺旋上的一圈所表示的活动。在每次演进的时候,都要考虑风险(第 26 章)。每个演进过程还要标记里程碑——沿着螺旋路径达到的工作产品和条件的结合体。

螺旋的第一圈一般开发出产品的规格说明,接下来开发产品的原型系统,并在每次迭代中逐步完善,开发不同的软件版本。螺旋的每圈都会跨过策划区域,此时,需调整项目计划,并根据交付后用户的反馈调整预算和进度。另外,项目经理还会调整完成软件开发需要迭代的次数。

其他过程模型在软件交付后就结束了。螺旋模型则不同,它应用在计算机软件的整个生命周期。因此,螺旋上的第一圈可能表示“概念开发项目”,它起始于螺旋的中心,经过多个迭代^②,直到概念开发的结束。如果这个概念将被开发成为实际的产品,那么该过程将继续沿着螺旋向外伸展,成为“新产品开发项目”。新产品将沿着螺旋通过一系列的迭代不断演进。最后,可以用一圈螺旋表示“产品提高项目”。本质上,当螺旋模型以这种方式进行下去的时候,它将永远保持可操作性,直到软件产品的生命周期结束。过程经常会处于休止状态,但每当有变更时,过程总能够在合适的入口点启动(如产品提高)。

螺旋模型是开发大型系统和软件的很实际的方法。由于软件随着过程的推进而变化,因此在每一个演进层次上,开发者和客户都可以更好地理解 and 应对风险。螺旋模型把原型作为降低风险的机制,更重要的是,开发

关键点 螺旋模型能运用在应用系统开发的整个生命周期,从概念开发到维护。

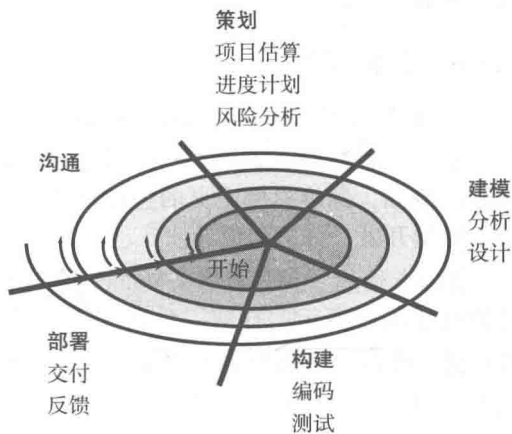


图 4-5 典型的螺旋模型

网络资源 可以在如下网址获得关于螺旋模型的有用信息: www.sei.cmu.edu/publications/documents/00.reports/00sr008.html。

建议 如果你的项目要求固定预算开发(通常不是一个好主意),那么螺旋模型会带来问题:每一圈完成的时候,都将重新计划和修改项目开销。

① 这里讨论的螺旋模型有别于 Boehm 提出的模型。原始的螺旋模型可参见 [Boe88]。关于 Boehm 的螺旋模型的更多更新的讨论可参见 [Boe98]。

② 沿轴指向中心的箭头区分开了部署和沟通两个区域,表明沿同一个螺旋路径存在潜在的局部迭代。

人员可以在产品演进的任何阶段使用原型方法。它保留了经典生命周期模型中系统逐步细化的方法，但是把它纳入一种迭代框架之中，这种迭代方式与真实世界更加吻合。螺旋模型要求在项目的所有阶段始终考虑技术风险，如果适当地应用该方法，就能够在风险变为难题之前将其化解。

与其他范型一样，螺旋模型也并不是包治百病的灵丹妙药。很难使客户（特别是以合同的形式）相信演进的方法是可控的。它依赖大量的风险评估专家来保证成功。如果存在较大的风险没有被发现和管理，就肯定会发生问题。

4.1.4 并发模型

并发开发模型（concurrent development model）有时也叫作并发工程，它允许软件团队表述本章所描述的任何过程模型中的迭代元素和并发元素。例如，螺旋模型定义的建模活动由以下一种或几种软件工动作完成：原型开发、分析和设计。[⊖]

图 4-6 给出了并发建模方法的一个例子。在某一特定时间，建模活动可能处于图中所示的任何一种状态[⊕]中。其他活动、动作或任务（如沟通或构建）可以用类似的方式表示。所有的软件工程活动同时存在并处于不同的状态。

引述 我今天只走这么远，只有明天才能为我指明方向。
Dave Matthews
Band

关键点 必须将项目计划看成是活的文档，必须经常对进度进行评估并考虑变更情况，从而对其进行修改。

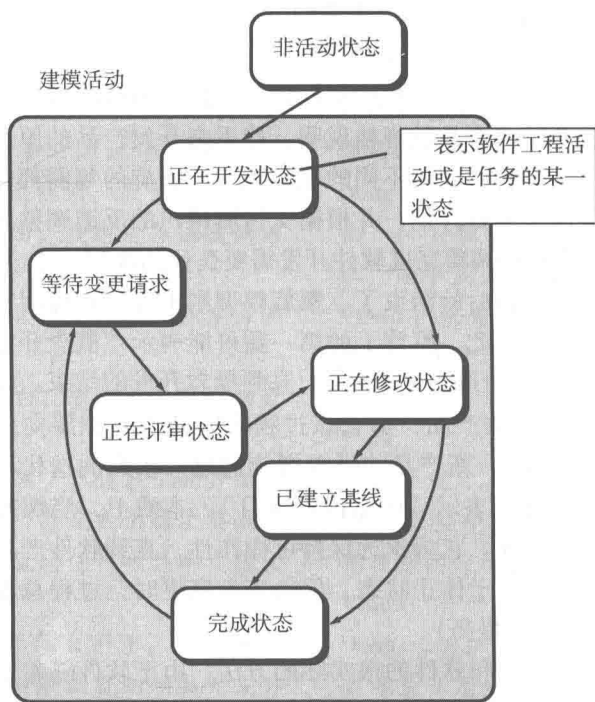


图 4-6 并发过程模型的一个元素

⊖ 需要注意的是，分析和设计都是很复杂的任务，需要进行大量的讨论。本书的第 2 部分将详细讨论这些问题。

⊕ 状态是外部可见的某种行为模式。

例如，在项目的早期，沟通活动（图中并未标明）完成了第一个迭代，停留在等待变更状态。建模活动（初始沟通完成后，一直停留在非活动状态）现在转换到正在开发状态。然而，如果客户要求必须完成需求变更，那么建模活动就会从正在开发状态转换到等待变更状态。

并发建模定义了一系列事件，这些事件将触发软件工程活动、动作或者任务的状态转换。例如，设计的早期阶段（建模活动期间发生的主要软件工程动作）发现了需求模型中的不一致性，于是产生了分析模型修正事件，该事件将触发需求分析动作从完成状态转换到等待变更状态。

并发建模可用于所有类型的软件开发，它能够提供更精确的项目当前状态图。它不是把软件工程活动、动作和任务局限在一个事件的序列，而是定义了一个过程网络。网络上每个活动、动作和任务与其他活动、动作和任务同时存在。过程网络中某一点产生的事件可以触发与每一个活动相关的状态的转换。

建议 并发模型更适合于不同软件工程团队共同开发的产品工程项目。

引述 确保你所在组织的每一个过程都有客户，如果没有客户，过程就会变成空中楼阁，失去目标。

V. Daniel Hunt

SafeHome 选择过程模型 第2部分

[场景] CPI 公司软件工程部会议室，该公司生产家用和商用消费类产品。

[人物] Lee Warren，工程经理；Doug Miller，软件工程经理；Vinod 和 Jamie，软件工团队成员。

[对话]

（Doug 介绍了一些可选的演化模型）

Jamie：我现在有了一些想法。增量模型挺有意义的。我很喜欢螺旋模型，听起来很实用。

Vinod：我赞成。我们交付一个增量产品，听取用户的反馈意见，再重新计划，然后交付另一个增量。这样做也符合产品的特性。我们能够迅速投入市场，然后在每个版本或者说在每个增量中添加功能。

Lee：等等，Doug，你的意思是说我们在螺旋的每一轮都重新生成计划？这样不好，我们需要一个计划，一个进度，然后严格遵守这个计划。

Doug：你的思想太陈旧了，Lee。就像他们说的，我们要现实。我认为，随着我们认识的深入和情况的变化来调整计划更好。这是一种更符合实际的方式。如果制定了不符合实际的计划，这个计划还有什么意义？

Lee（皱眉）：我同意这种看法，可是高管人员不喜欢这种方式，他们喜欢确定的计划。

Doug（笑）：老兄，你应该给他们上一课。

4.1.5 演化过程的最终评述

我们注意到，现代计算机软件总是在持续改变，这些变更通常要求在非常短的期限内实现，并且要充分满足客户—用户的要求。许多情况下，及时投入市场是最重要的管理要求。如果市场时间错过了，软件项目自身可能会变得毫无意义。^①

① 需要注意的是，尽管及时投入市场很重要，但最早进入市场并不保证一定成功。事实上，许多非常成功的软件是第二个或是第三个进入市场的（他们的成功在于吸取了前人的教训）。

演化过程模型就是为了解决上述问题的，但是，作为一类通用的过程模型，它们也有缺点。Nogueira 和他的同事对此概括如下 [Nog00]：

尽管演化软件过程毫无疑问具有一定的优势，但我们还是有一些担忧。首先，由于构建产品需要的周期数目不确定，原型开发（和其他更加复杂的演化过程）给项目计划带来了困难……

其次，演化软件过程没有确定演化的最快速度。如果演化的速度太快，完全没有间歇时间，项目肯定会陷入混乱；反之，如果演化速度太慢，则会影响生产率……

再次，演化软件过程应该侧重于灵活性和可延展性，而不是高质量。这种说法听起来很惊人。

确实，一个强调灵活性、可扩展性和开发速度而不是高质量的软件过程听起来令人震惊。可是，很多广为人们尊重的软件工程专家都这样建议（例如 [You95]、[Bac97]）。

演化模型的初衷是采用迭代或者增量的方式开发高质量软件^①。可是，用演化模型也可以做到强调灵活性、可延展性和开发速度。软件团队及其经理所面临的挑战就是在这些严格的项目、产品参数与客户（软件质量的最终仲裁者）满意度之间找到一个合理的平衡点。

提问 演化过程模型具有哪些不为人知的弱点？

4.2 专用过程模型

专用过程模型具有前面章节中提到的传统过程模型的一些特点，但是，专用过程模型往往应用面较窄且较专一，只适用于某些特定的软件工程方法。^②

4.2.1 基于构件的开发

商业现货（Commercial Off-The-Shelf, COTS）软件构件由厂家作为产品供应，通过良好定义的接口提供特定的功能，这些构件能够集成到正在构建的软件中。基于构件的开发模型（component-based development model）具有许多螺旋模型的特点。它本质上是演化模型 [Nie92]，需要以迭代方式构建软件。不同之处在于，基于构件的开发模型采用预先打包的软件构件来开发应用系统。

网络资源 基于构件开发的相关信息可以参见 www.cbd-hq.com。

建模和构建活动开始于识别可选构件。这些构件有些设计成传统的软件模块，有些设计成面向对象的类或类包^③。若不考虑构件的开发技术，则基于构件开发模型由以下步骤组成（采用演化方法）：

1. 对于该问题的应用领域研究和评估可用的基于构件的产品。
2. 考虑构件集成的问题。
3. 设计软件架构以容纳这些构件。
4. 将构件集成到架构中。
5. 进行充分的测试以保证功能正常。

① 在这里，软件质量的含义非常广泛，不仅包括客户满意度，还包括本书第二部分讲的各种技术指标。

② 在某些情况下，这些专用过程模型也许应该更确切地称为技术的集合或“方法论”，是为了实现某一特定的软件开发目标而制定的。但它们确实也提出了一种过程。

③ 附录 2 讨论了面向对象概念，该概念的使用贯穿了本书第二部分。在这里，类封装了一组数据以及处理数据的过程。类包是一组共同产生某种结果的相关类的集合。

基于构件的开发模型能够使软件复用，从而为软件工程师带来极大收益。如果构件复用已经成为你所在的软件工程团队文化的一部分，那么将会缩短开发周期并减少项目开发费用。基于构件的软件开发将在第13章进行详细讨论。

4.2.2 形式化方法模型

形式化方法模型（formal methods model）的主要活动是生成计算机软件形式化的数学规格说明。形式化方法使软件开发人员可以应用严格的数学符号来说明、开发和验证基于计算机的系统。这种方法的一个变形是净室软件工程（cleanroom software engineering）[Mil87, Dye92]，这一软件工程方法目前已应用于一些软件开发机构。

形式化方法提供了一种机制，使得在软件开发中可以避免一些问题，而这些问题在使用其他软件工程模型时是难以解决的。使用形式化方法时，歧义性问题、不完整问题、不一致问题等都能够更容易地被发现和改正——不是依靠特定的评审，而是应用数学分析的方法。在设计阶段，形式化方法是程序验证的基础，使软件开发人员能够发现和改正一些常常被忽略的问题。

虽然形式化方法不是一种主流的方法，但它的意义在于可以提供无缺陷的软件。尽管如此，人们还是对在商业环境中应用形式化方法有怀疑，这表现在：

- 目前，形式化模型开发非常耗时，成本也很高。
- 只有极少数程序员具有应用形式化方法的背景，因此需要大量的培训。
- 对于技术水平不高的客户，很难用这种模型进行沟通。

尽管有这些疑虑，但软件开发中还是有很多形式化方法的追随者，比如有人用其开发那些高度关注安全的软件（如飞行器和医疗设施），或者开发那些一旦出错就将导致重大经济损失的软件。

提问 既然形式化方法能够保证软件的正确性，为什么没有被广泛应用呢？

4.2.3 面向方面的软件开发

不论选择什么软件过程，复杂软件都无一例外地实现了一套局部化的特性、功能和信息内容。这些局部的软件特性被做成构件（例如面向对象的类），然后在系统架构中使用。随着现代计算机系统变得更加复杂，某些关注点——客户需要的属性或者技术兴趣点——已经体现在整个架构设计中。有些关注点是系统的高层属性（例如安全性、容错能力），另一些关注点影响了系统的功能（例如商业规则的应用），还有一些关注点是系统性的（例如任务同步或内存管理）。

如果某个关注点涉及系统多个方面的功能、特性和信息，那么这些关注点通常称为横切关注点（crosscutting concern）。方面的需求（aspectual requirement）定义那些对整个软件体系结构产生影响的横切关注点。面向方面的软件开发（Aspect-Oriented Software Development, AOSD）通常称为面向方面编程（Aspect-Oriented Programming, AOP）或者面向方面构件工程（Aspect-Oriented Component Engineering, AOCE）[Gru02]，它是相对较新的一种软件工程模型，为定义、说明、设计和构建方面（aspect）提供过程和方法——“是对横切关注点进行局部表示的一种机制，超越了子程序和继承方法”[Elr01]。

网络资源 关于AOP的资源和信息可以参见aosd.net。

关键点 AOSD定义了“方面”，表示用户跨越多个系统功能、特性和信息的关注点。

与众不同的面向方面的过程还不成熟。尽管如此,这种过程模型看似具备了演化模型和并发过程模型的共同特点。演化模型适合定义和构建方面;而并发开发的并行特点很重要,因为方面是独立于局部的软件构件开发的,并且对这些构件的开发有直接影响。因此,在构建方面和构件的过程活动之间建立起异步的通信非常重要。

关于面向方面的软件开发最好查阅相关专著中的详细讨论。感兴趣的读者可以参看[Ras11]、[Saf08]、[Cla05]、[Fil05]、[Jac04]和[Gra03]。

软件工具 过程管理

[目标] 辅助定义、执行和管理传统过程模型。

[机制] 过程管理工具帮助软件组织或团队定义完整的软件过程模型(框架活动、动作、任务、质量保证检查点、里程碑和工作产品)。而且,该工具为软件工程师的技术工作提供路线图,为经理们跟踪和控制软件过程提供模板。

[代表性工具]^①

- GDPA。一个研究性的过程定义工具包,该工具包由德国的 Bremen 大学(www.informatik.uni-bremen.de/uniform/gdpa/home.htm)开发,它提供了大量的过程

建模和管理功能。

- ALM Studio。由 Kovair 公司(<http://www.kovair.com/>)开发的一套工具包,用于过程定义、需求管理、问题解决、项目策划和跟踪。
- ProVision BPMx。由 OpenText(<http://bps.opentext.com>)开发,它提供了很多代表性工具,可以辅助过程定义和工作流自动化。

以下网站提供了很多与软件过程相关的各种很有价值的工具:www.computer.org/portal/web/swbok/html/ch10。

4.3 统一过程

Ivar Jacobson、Grady Booch 和 James Rumbaugh[Jac99]在他们关于统一过程(unified process)的影响深远的著作中,讨论了关于有必要建立一种“用例驱动,以架构为核心,迭代并且增量”的软件过程的问题,并阐述如下:

当前,软件朝着更大、更复杂的系统发展。部分原因在于计算机的计算能力逐年递增,使得人们对其的期望值增大。另一方面,还是受 Internet 应用膨胀的影响,促使各类信息交换……我们从软件版本的提升中学会了如何改进产品,使我们对更复杂软件的胃口越来越大。同时希望软件更好地满足我们的需要,结果导致软件更加复杂。简而言之,我们想要的越来越多。

在某种程度上,统一过程尝试着从传统的软件过程中挖掘最好的特征和性质,但是以敏捷软件开发(第7章)中许多最好的原则来实现。统一过程认识到与客户沟通以及从用户的角度描述系统(即用例)^②并保持该描述的一致性的的重要性。它强调软件体系结构的重要

① 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。大多数情况下,工具的名字由各自的开发者注册为商标。

② 用例(use case)(第7章)是一种文字描述或模板,从用户的角度描述系统功能和特性。用例由用户来写,并作为创建更为复杂的分析模型的基础。

作用，并“帮助架构师专注于正确的目标，例如可理解性、对未来变更的可适应性以及复用”[Jac99]。它建立了迭代的、增量的过程流，提供了演进的特性，这对现代软件开发非常重要。

4.3.1 统一过程的简史

20 世纪 90 年代早期，James Rumbaugh [Rum91]、Grady Booch [Boo94] 和 Ivar Jacobson [Jac92] 开始研究“统一方法”，他们的目标是结合各自面向对象分析和设计方法中最好的特点，并吸收其他面向对象模型专家提出的其他特点（例如 [Wir90]）。他们的成果就是 UML——统一建模语言（unified modeling language），这种语言包含了大量用于面向对象系统建模和开发的符号。到了 1997 年，UML 已经变成了面向对象软件开发的行业标准。

UML 作为需求模型和设计模型的表示方式，其应用贯穿本书第二部分。附录 1 针对不熟悉 UML 基本概念和建模规则的人给出了介绍性的指导。有关 UML 的全面介绍可以参考有关 UML 的材料，附录 1 中列出了相关书籍。

4.3.2 统一过程的阶段^①

在第 3 章，我们讨论了 5 种通用的框架活动，并认为它们可以用来描述任何软件过程模型。统一过程也不例外。图 4-7 描述了统一过程（Unified Process, UP）的“阶段”，并将他们与第 1 章及本章前面部分讨论的通用活动进行了对照。

UP 的起始阶段（inception phase）包括客户沟通和策划活动。通过与利益相关者协作定义软件的业务需求，提出系统大致的架构，并制定开发计划以保证项目开发具有迭代和增量的特性。该阶段识别基本的业务需求，并初步用例（第 7 章）描述每一类用户所需要的主要特性和功能。此时的体系结构仅是主要子系统及其功能、特性的试探性概括。随后，体系结构将被细化和扩充成为一组模型，以描述系统的不同视图。策划阶段将识别各种资源，评估主要风险，制定进度计划，并为其在软件增量开发的各个阶段中的应用建立基础。

细化阶段（Elaboration Phase）包括沟通和通用过程模型的建模活动（图 4-7）。细化阶段扩展了初始阶段定义的用例，并扩展了体系结构以包括软件的五种视图——用例模型、需求模型、设计模型、实现模型和部署模型。在某些情况下，细化阶段建立了一个“可执行的体系结构基线”[Arl02]，这是建立可执行系统的“第一步”（first cut）^②。体系结构基线证明了体系结构的可实现性，但没有提供系统使用时所需的所有功能和特性。另

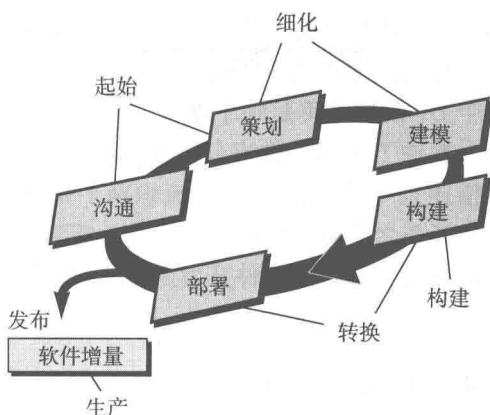


图 4-7 统一过程

关键点 UP 阶段的目的与本书中定义的通用框架活动的目的是类似的。

① 统一过程有时也用 Rational 公司（后来被 IBM 收购）的名字命名，称为 Rational 统一过程（Rational Unified Process, RUP），Rational 公司是早期开发和细化该统一过程的主要投资方，并建立了支持该过程的完整环境（工具及技术）。

② 需要指出的是，体系结构基线不是原型系统，因为它并不会被抛弃，而是在下一个 UP 阶段进一步充实。

外,在细化的最终阶段将评审项目计划以确保项目的范围、风险和交付日期的合理性。该阶段通常要对项目计划进行修订。

UP 的构建阶段 (construction phase) 与通用软件过程中的构建活动相同。构建阶段采用体系结构模型作为输入,开发或是获取软件构件,使得最终用户能够操作用例。为达到上述目的,要对在细化阶段开始的需求模型和设计模型加以完善,以反映出软件增量的最终版本。软件增量(例如发布的版本)所要求的必须具备的特性和功能在源代码中实现。随着构件的实现,对每一个构件设计并实施单元测试^①。另外,还实施了其他集成活动(构件组装和集成测试)。用例用于导出一组验收测试,以便在下一个 UP 阶段开始前执行。

网络资源 敏捷开发中统一过程的有趣讨论参见 www.ambysoft.com/unifiedprocess/agileUP.html。

UP 的转换阶段 (transition phase) 包括通用构建活动的后期阶段以及通用部署(交付和反馈)活动的第一部分。软件被提交给最终用户进行 Beta 测试,用户反馈报告缺陷及必要的变更。另外,软件开发团队创建系统发布所必需的支持信息(如用户手册、问题解决指南及安装步骤)。在转换阶段结束时,软件增量成为可用的发布版本。

UP 的生产阶段 (production phase) 与通用过程的部署活动一致。在该阶段,对持续使用的软件进行监控,提供运行环境(基础设施)的支持,提交并评估缺陷报告和变更请求。

有可能在构建、转换和生产阶段的同时,下一个软件增量的工作已经开始。这就意味着五个 UP 阶段并不是顺序进行,而是阶段性地并发进行。

软件工程的工作流分布在所有 UP 阶段。在 UP 中,工作流类似于任务集(参见第 3 章的描述)。也就是说,工作流识别了完成一个重要的软件工程活动的必要任务,以及在成功完成任务之后所产生的工作产品。需要注意的是,并不是工作流所识别的每一个任务都在所有的项目中应用。软件开发团队应根据各自的需要适当调整过程(动作、任务、子任务及工作产品)。

4.4 产品和过程

如果过程很薄弱,则最终产品必将受到影响。但是对分过程的过分依赖也是很危险的。Margaret Davis[Dav95a] 在多年前写的一篇简短的文章里对产品和过程的双重性进行了以下评述:

大约每十年或五年,软件界都会对“问题”重新定义,其重点由产品问题转向了过程问题。因此,我们逐步采纳了结构化程序设计语言(产品)、结构化分析方法(过程)和数据封装(产品),到现在重点是卡内基·梅隆大学软件工程研究所提出的能力成熟度模型(过程)(随后逐步采纳面向对象方法和敏捷软件开发)。

钟摆的自然趋势是停留在两个极端的中点,与之类似,软件界的关注点也不断地摆动,当上一次摆动失败时,就会有新的力量加入,促使它摆向另一个方向。这些摆动是非常有害的,因为它们可能从根本上改变了工作内容及工作方法,使软件工程实践人员陷入混乱。而且这些摆动并没有解决问题,只是把产品和过程分裂开来而不是作为辩证统一的一体,因此注定要失败。

这种二象性在科学界早有先例,当某一个理论不能对观测到的相互矛盾的结果做出合理

^① 有关软件测试(包括单元测试)的深入讨论参见第 17 ~ 19 章。

解释时,就会出现二象性理论。由 Louis de Broglie 于 20 世纪 20 年代提出的光的波粒二象性就是一个很好的例子。我相信,我们对软件组成部分和开发过程的观测证明了软件具有过程和产品的二象性。如果仅仅将软件看作一个过程或是一个产品,那就永远都不能正确地理解软件,包括其背景、应用、意义和价值。

所有的人类活动都可以看成是一个过程,我们每一个人都从这些活动中获得对自我价值的认识,这些活动所产生的结果可以被许多人反复地在不同的情况下使用。也就是说,我们是从我们自己或是他人对我们产品的复用中得到满足的。

因此,将复用目标融入软件开发,这不仅潜在地增加了软件专业人员从工作中获得的满足感,也增加了接受“产品和过程二象性”这一观点的紧迫性。对于一个可复用的部件,如果仅仅从产品或是仅仅从过程的角度考虑,都不利于软件开发,这种片面的观点或者影响了人们对产品的应用环境 and 应用方法的认识,或者忽略了该产品还可以作为其他开发活动的输入这一事实。因此,片面地强调某一方面的观点会极大地降低软件复用的可能性,也会大大减少工作的成就感。

正如从最终产品获得满足一样,人们在创造性的过程中得到了同样的(甚至更大的)成就感。艺术家不仅仅对装裱好的画卷感到高兴,更在每一笔绘画的过程中享受乐趣;作家不仅欣赏出版的书籍,更为每一个苦思得到的比喻而欣喜。一个具有创造性的专业软件人员也应该从过程中获得满足,其程度不亚于最终的产品。产品和过程的二象性已经成为保留推动软件工程不断进步的创造性人才的一个重要因素。

习题与思考题

- 4.1 详细描述三个适于采用瀑布模型的软件项目。
- 4.2 详细描述三个适于采用原型模型的软件项目。
- 4.3 如果将原型变成一个可发布的系统或者产品,应该如何调整过程?
- 4.4 详细描述三个适于采用增量模型的软件项目。
- 4.5 当沿着螺旋过程流发展的时候,你对正在开发或者维护的软件的看法是什么?
- 4.6 可以合用几种过程模型吗?如果可以,举例说明。
- 4.7 并发过程模型定义了一套“状态”,用你自己的话描述一下这些状态表示什么,并指出他们在并发过程模型中的作用。
- 4.8 开发质量“足够好”的软件,其优点和缺点是什么?也就是说,当我们追求开发速度胜过产品质量的时候,会产生什么后果?
- 4.9 详细描述三个适于采用基于构件模型的软件项目。
- 4.10 我们可以证明一个软件构件甚至整个程序的正确性,可是为什么并不是每个人都这样做?
- 4.11 统一过程和 UML 是同一概念吗?解释你的答案。

扩展阅读与信息资源

第 2 章的扩展阅读部分提到的大部分资料都详细地介绍了传统过程模型。

Cynkovic 和 Larsson (《Building Reliable Component-Based Systems》, Addison-Wesley, 2002) 以及 Heineman 和 Council (《Component-Based Software Engineering》, Addison-Wesley, 2001) 描述了实现基于构件系统的过程需求。Jacobson 和 Ng (《Aspect-Oriented Software Development with Use Cases》, Addison-Wesley, 2005) 以及 Filman 和他的同事 (《Aspect-Oriented Software Development》, Addison-Wesley, 2004) 讨论了面向方面过程的独特性质。Monin 和 Hinchey (《Understanding Formal Methods》,

Springer, 2003) 提供了有价值的介绍, Baco 和他的同事 (《Formal Methods》, Springer, 2009) 讨论了目前的最新水平和新方向。

Kenett 和 Baker (《Software Process Quality : Management and Control》, Marcel Dekker, 1999) 以及 Chrissis、Konrad 和 Shrum (《CMMI for Development : Guidelines for Process Integration and Product Improvement》, 3rd ed., Addison-Wesley, 2011) 考虑了高质量的管理和过程设计是如何相互影响的。

除了 Jacobson、Rumbaugh 和 Booch 的有关统一过程的书籍 [Jac99] 以外, Shuja 和 Krebs (《IBM Rational Unified Process Reference and Certification Guide》, IRM Press, 2008)、Arlow 和 Neustadt (《UML 2 and the Unified Process》, Addison-Wesley, 2005)、Kroll 和 Kruchten (《The Rational Unified Process Made Easy》, Addison-Wesley, 2003) 以及 Farve (《UML and the Unified Process》, IRM Press, 2003) 等人的书籍提供了很好的补充信息。Gibbs (《Project Management with the IBM Rational Unified Process》, IBM Press, 2006) 讨论了统一过程中的项目管理问题。Dennis、Wixom 和 Tegarden (《System Analysis and Design with UML》, 4th ed., Wiley, 2012) 解决涉及 UP 的编程和业务过程建模问题。

网上有大量关于软件过程模型的信息源, 与软件过程有关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

敏捷开发

要点浏览

概念：敏捷软件工程是哲学理念和一系列开发指南的综合。这种哲学理念推崇：让客户满意且尽早的增量发布；小而高度自主的项目团队；非正式的方法；最小化软件工作产品以及整体精简开发。开发的指导方针强调超越分析和设计（尽管并不排斥这类活动）的发布，以及开发人员和客户之间主动和持续的沟通。

人员：软件工程师和其他项目利益相关者（经理、客户、最终用户）共同组成敏捷开发团队，这个团队是自我组织的并掌握着自己的命运。敏捷团队鼓励所有参与人员之间的交流与合作。

重要性：孕育着基于计算机的系统和软件产品的现代商业环境正以飞快的节奏不断变化着。敏捷软件工程提出了针对特定类型软件和软件项目的不同于传统软

件工程的合理方案。事实证明，这一方法可以快速交付成功的系统。

步骤：对敏捷开发的恰当称呼应当是“软件工程精简版”，它保留了基本的框架活动：客户沟通、策划、建模、构建和部署，但将其缩减到一个推动项目组朝着构建和交付发展的最小任务集（有人认为这种方法是牺牲问题分析和方案设计为代价而实现的）。

工作产品：客户和软件工程师有着共同的观点——唯一真正重要的工作产品是在合适的时间交付给客户的可运行软件增量。

质量保证措施：如果敏捷团队认为过程可行，并且开发出的可交付软件增量能使客户满意，则软件质量就是没有问题的。

2001年，Kent Beck和其他16位知名软件开发者和软件工程作家以及软件咨询师[Bec01]（被称为敏捷联盟）共同签署了“敏捷软件开发宣言”。该宣言声明：

我们正在通过亲身实践以及帮助他人实践的方式来揭示更好的软件开发之路，通过这项工作，我们认识到：

- 个人和他们之间的交流胜过了开发过程和工具
- 可运行的软件胜过了宽泛的文档
- 客户合作胜过了合同谈判
- 对变更的良好响应胜过了按部就班地遵循计划

也就是说，虽然上述右边的各项很有价值，但我们认为左边的各项具有更大的价值。

一份宣言通常和一场即将发生的破旧立新的政治运动（期望好转）相关联。从某些方面来讲，敏捷开发确实是这样一场运动。

关键概念

验收测试
敏捷联盟
敏捷过程
敏捷统一过程
敏捷
敏捷原则
变更成本
动态系统开发方法 (DSDM)
极限编程 (XP)
工业 XP
结对编程
敏捷开发战略
项目速度

虽然多年来大家一直都在使用着指导敏捷开发的基本思想，但真正将它们凝聚到一场“运动”中还不到二十年。从本质上讲，敏捷方法^①是为了克服传统软件工程中认识和实践的弱点而形成的。敏捷开发可以带来多方面的好处，但它并不适用于所有的项目、所有的产品、所有的人和所有的情况。它并不完全对立于传统软件工程实践，也不能作为超越一切的哲学理念而用于所有软件工作。

在现代经济生活中，通常很难甚至无法预测一个基于计算机的系统（如移动 App）如何随时间推移而演化。市场情况变化迅速，最终用户需求不断变更，新的竞争威胁毫无征兆地出现。在很多情况下，在项目开始之前，我们无法充分定义需求。因此，我们必须足够敏捷地去响应不断变化、无法确定的商业环境。

不确定性意味着变更，而变更意味着付出昂贵的成本，特别是在其失去控制或疏于管理的情况下，为这种变更而付出的成本费用是昂贵的。而敏捷方法最具强制性的特点之一就是它能够通过软件过程来降低由变更所引起的代价。

难道说认识到现实的挑战，我们就完全抛弃那些有价值的软件工程原理、概念、方法和工具吗？绝对不是。和其他所有工程学科一样，软件工程也在持续发展着，我们可以通过改进软件工程本身来适应敏捷带来的挑战。

Alistair Cockburn[Coc02] 在他那本发人深省的敏捷软件开发著作中，论证了本书第 4 章介绍的惯例过程模型中存在的主要缺陷：忘记了开发计算机软件的人员的弱点。软件工程师不是机器人，许多软件工程师在工作方式上有很大差别，在技能水平、主动性、服从性、一致性和责任心方面也有巨大差异。有些人可以通过书面方式很好地沟通，而有些人则不行。Cockburn 论证说：过程模型可以“利用纪律或者宽容来处理人的共同弱点”，因而大多数惯例过程模型选择了纪律。他还指出：“不能始终一致地做同一件事是人性的弱点，因而高度纪律性的方法学非常脆弱。”

要想让过程模型可用，要么必须提供实际可行的机制来维持必要的纪律，要么必须“宽容”地对待软件工程师。显而易见，“宽容”更易于被接受和保持，但是（正如 Cockburn 所认同）可能效率低下。正像人生中的大多数事情一样，必须权衡利弊。

5.1 什么是敏捷

在软件工程师工作这个环境下，什么是敏捷？Ivar Jacobson[Jac02a] 给出一个非常有用的论述：

敏捷已经成为当今描述现代软件过程的时髦用词。每个人都是敏捷的。敏捷团队是能够适当响应变更的灵活团队。变更就是软件开发本身，软件构建有变更、团队成员在变更、使用新技术会带来变更，各种变更都会对开发的软件产品以及项目本身造成影响。我们必须接受“支持变更”的思想，它应当根植于软件开发中的每一件事中，因为它是软件的心脏与灵魂。敏捷团队意识到软件是团队中所有人共同开发完成的，这些人的个人技能和合作能力是项目成功的关键所在。

关键概念

重构
Scrum
spike 解决方案
XP 故事

关键点

敏捷开发并不意味着不创建任何文件，但其仅在开发过程后期创建文件。

引述 敏捷：1，其他：0。

Tom DeMarco

^① 敏捷方法有时候也被称为轻量级方法或精简方法。

在 Jacobson 看来,普遍存在的变更是敏捷的基本动力,软件工程师必须加快步伐以适应 Jacobson 所描述的快速变更。

但是,敏捷不仅仅是有效地响应变更,它还包含着对本章开头的宣言中提及哲学观念的信奉。它鼓励能够使沟通(团队成员之间、技术和商务人员之间、软件工程师和经理之间)更便利的团队结构和协作态度。它强调可运行软件的快速交付而不那么看重中间产品(这并不总是好事情);它将客户作为开发团队的一部分开展工作,以消除持续、普遍存在于多数软件项目中的“区分我们和他们”的态度;它意识到在不确定的世界里计划是有局限性的,项目计划必须是可以灵活调整的。

敏捷可以应用于任何软件过程。但是,为了实现这一目标,非常重要的一点是:过程的设计应使项目团队适应于任务,并且使任务流水线化,在了解敏捷开发方法的流动性的前提下进行计划的制定,保留最重要的工作产品并使其保持简洁,强调这样一个增量交付策略——根据具体的产品类型和运行环境,尽可能快地将可工作的软件交付给客户。

建议 不要误解,以为敏捷会给你制定出解决方案的许可证。过程还是需要的,而规范是必不可少的。

5.2 敏捷及变更成本

软件开发的传统方法中(有几十年的开发经验作支持),变更成本随着计划的进展成非线性增长(图 5-1,黑色曲线)。这种方法在软件开发团队收集需求时(在项目的早期)相对容易适应变更。应用场景需要修改,功能表应该扩充,或者书面说明书需要编辑。这项工作的费用是最小的,所需的时间不会严重影响项目的结果。但是,如果我们在经过数月的开发时间之后将会怎么样?团队正在进行确认测试(也许是在项目后期的某个活动中),这时一个重要的利益相关者要求变更一个主要功能。这一变更需要对软件的体系结构设计进行修改,包括设计和构建三个新构件、修改另外五个构件、设计新的测试等。成本会迅速升级,为了保证变更不会引起非预期的副作用,所需的时间和费用都是非常可观的。

敏捷的拥护者(例如 [Bec00]、[Amb04])认为,一个设计良好的敏捷过程“拉平”了变更曲线(图 5-1,灰色曲线),使软件开发团队在没有超常规的时间和费用影响的情况下,在软件项目后期能够适应各种变更。大家已经学习过,敏捷过程包括增量交付。当增量交付与其他敏捷实践结合时,例如连续单元测试及结对编程(在本章后面讨论),引起变更的费用会衰减。虽然关于拉平曲线程度的讨论仍然在进行,但是证据表明 [Coc01a],变更成本显著降低。

引述 敏捷是动态的、针对特定内容的、主动应对变更及面向成长的。

Steven Goldman
et al.

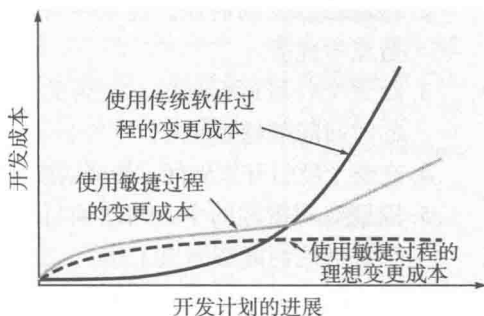


图 5-1 变更成本是开发时间的函数

5.3 什么是敏捷过程

任何敏捷软件过程的特征都是以某种方式提出若干关键假设 [Fow02], 这些假设可适用于大多数软件项目。

关键点 敏捷过程能够降低变更的成本是因为软件产品以增量方式发布,而且在增量内部变更能得到较好的控制。

1. 提前预测哪些需求是稳定的以及哪些需求会变更是非常困难的。同样, 预测项目进行客户优先级的变更也很困难。
2. 对很多软件来说, 设计和构建是交错进行的。也就是两种活动应当顺序开展以保证通过构建实施来验证设计模型, 而在通过构建验证之前很难估计应该设计到什么程度。
3. 分析、设计、构建和测试并不像我们所设想的那么容易预测(从制定计划的角度来看)。

网络资源 有关敏捷过程的综合文集可在以下网站找到: <http://www.agilemodeling.com>。

给出这三个假设, 同时也就提出一个重要的问题: 如何建立能解决不可预测性的过程? 正如前文所述, 答案就在于过程的可适应性(对于快速变更的项目和技术条件)。因此, 敏捷过程必须具有可适应性。

但是原地踏步式的连续适应性变更收效甚微, 因而, 敏捷软件过程必须增量地适应。为了达到这一目的, 敏捷团队需要客户的反馈(以做出正确的适应性改变), 可执行原型或部分实现的可运行系统是客户反馈的最有效媒介。因此, 应当使用增量式开发策略, 必须在很短的时间间隔内交付软件增量(可执行原型或部分实现的可运行系统)来适应(不可预测的)变更的步伐。这种迭代方法允许客户: 周期性地评价软件增量, 向软件项目组提出必要的反馈, 影响为适应反馈而对过程进行的适应性修改。

5.3.1 敏捷原则

敏捷联盟(参见[Agi03]、[Fow03])为希望达到敏捷的人们定义了12条原则:

关键点 尽管敏捷过程支持变更, 但检查变更的原因仍然是重要的。

1. 我们最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。
2. 即使在开发的后期, 也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。
3. 经常交付可运行软件, 交付的间隔可以从几个星期到几个月, 交付的时间间隔越短越好。
4. 在整个项目开发期间, 业务人员和开发人员必须天天都在一起工作。
5. 围绕有积极性的个人构建项目。给他们提供所需的环境和支持, 并且信任他们能够完成工作。
6. 在团队内部, 最富有效果和效率的信息传递方法是面对面交谈。
7. 可运行软件是进度的首要度量标准。
8. 敏捷过程提倡可持续的开发速度。责任人(sponsor)、开发者和用户应该能够长期保持稳定的开发速度。
9. 不断地关注优秀的技能和好的设计会增强敏捷能力。
10. 简单——使不必做的工作最大化的艺术——是必要的。
11. 最好的架构、需求和设计出自于自组织团队。
12. 每隔一定时间, 团队会反省如何才能更有效地工作, 并相应调整自己的行为。

建议 可运行的软件至关重要, 但不可忘记还必须使软件具有包括可靠性、可用性以及可维护性在内的各种质量属性。

并不是每一个敏捷过程模型都同等使用这12项原则, 一些模型可以选择忽略(或至少淡化)一项或多项原则的重要性。然而, 上述原则定义了一种敏捷精神, 这种精神贯穿于本章提出的每一个过程模型。

5.3.2 敏捷开发战略

与较为传统的软件工程过程相反,敏捷软件开发在优越性和适用性方面存在着许多(有时是激烈的)争议。在表达对敏捷拥护者阵营(“敏捷者”)的感想时,Jim Highsmith [Hig02a] (开玩笑地)说明了他那颇为极端的观点:“传统方法学家陷入了误区,乐于生产完美的文档而不是满足业务需要的可运行系统。”而在表述对传统软件工程阵营的立场时,他则给出完全相反的观点(同样是玩笑性质的):“轻便级方法或者说敏捷方法学家是一群自以为了不起的黑客,他们妄图将其手中的软件玩具放大到企业级软件而制造出一系列轰动。”

建议 你不必在敏捷和软件工程之间做选择。只需要定义敏捷的软件工程方法。

像所有的软件技术争论一样,这场方法学之争有滑向派别之战的危险。一旦争吵发生,理智的思考就消失了,信仰而不是事实主导着各种决定。

没有人反对敏捷。而真正的问题在于“什么是最佳实现途径”。同等重要的还有,如何构建满足用户当前需要的软件,同时展示具有能满足客户长期需求的扩展能力?

对这两个问题还没有绝对正确的答案。即便在敏捷学派内部,针对敏捷问题,也提出了很多有细微差异的过程模型(5.4节),每个模型内有一组“想法”(敏捷者们不愿称其为“工作任务”),显现出和传统软件工程的显著差异。同时,许多敏捷概念是从优秀的软件工程概念简单地修正而来的。归根结底:兼顾两派的优点则双方都能得到很多利益,而相互诽谤只会两败俱伤。

感兴趣的读者可以参看[Hig01]、[Hig02a]和[DeM02],这些文献针对很多重要技术和方针问题给出了饶有趣味的总结。

5.4 极限编程

为了更详尽地说明敏捷过程,在此提供一个称为极限编程(eXtreme Programming, XP)的论述,它是敏捷软件开发中使用最广泛的一种方法。虽然极限编程相关的思想和方法最早出现于20世纪80年代后期,但具有开创意义的著作由Kent Beck撰写[Bec04a]。近年来,XP的变种——工业XP(IXP)被提了出来,目标是在大型组织内部使用敏捷过程[Ker05]。

网络资源 屡获殊荣的包括XP处理模块的“过程模拟对策”可参见<http://www.ics.uci.edu/~emilyo/SimSE/downloads.html>。

5.4.1 极限编程过程

XP使用面向对象方法作为推荐的开发范型(附录2),它包含了策划、设计、编码和测试4个框架活动的规则和实践。图5-2描述了XP过程,并指出与各框架活动相关的关键概念和任务。以下段落将概括XP关键的活动。

提问 XP中的“故事”是什么?

策划。策划活动(也称为策划比赛)开始于倾听,这是一个需求收集活动,该活动要使XP团队技术成员理解软件的商业背景,充分感受要求的输出和主要特性及主要功能。倾听产生一系列“故事”(也称为用户故事),描述将要开发的软件所需要的输出、特性以及功能。每个故事(类似于第7章讲述的用例)由客户书写并置于一张索引卡上,客户根据对应特征或功能的综合业务价值标明故事的权值(即优先级)^①。XP团队成员评

网络资源 XP“策划比赛”可以参见<http://csis.pace.edu/~bergin/xp/planninggame.html>。

① 一个故事的权值也可能取决于其他故事的存在。

估每一个故事，并给出以开发周数为度量单位的成本。如果某个故事的成本超过了3个开发周，则将请客户把该故事进一步细分，重新赋予权值并计算成本。重要的是应注意到新故事可以在任何时刻书写。

客户和XP团队共同决定如何将故事分组，并置于XP团队将要开发的下一个发行版本（下一个软件增量）中。一旦认可对下一个发布版本的基本承诺（就包括的故事、交付日期和其他项目事项），XP团队将以下述三种方式之一对有待开发的故事进行排序：（1）所有选定故事将（在几周之内）尽快实现；（2）具有最高价值的故事将移到进度表的前面并首先实现；（3）高风险故事将首先实现。

项目的第一个发行版本（也称为一个软件增量）交付之后，XP团队计算项目的速度。简而言之，项目速度是第一个发行版本中实现的客户故事个数。项目速度将用于：（1）帮助估计后续发行版本的发布日期和进度安排；（2）确定是否对整个开发项目中的所有故事有过分承诺。一旦发生过分承诺，则调整软件发行版本的内容或者改变最终交付日期。

在开发过程中，客户可以增加故事、改变故事的权值、分解或者去掉故事。接下来由XP团队重新考虑所有剩余的发行版本并相应修改计划。

设计。XP设计严格遵循KIS（Keep It Simple，保持简洁）原则，即使用简单的设计，而不是复杂的表述。另外，设计为故事提供恰好可实现的指导，而不鼓励额外功能性设计（因开发者假定以后会用到）^①。

XP鼓励使用CRC卡（第9章）作为在面向对象环境中考虑软件的有效机制。CRC（类-职责-协作者）卡确定和组织与当前软件增量相关的面向对象的类^②。XP团队使用类似于第9章描述的过程来管理设计工作。CRC卡也是作为XP过程一部分的唯一设计工作产品。

如果在某个故事设计中碰到困难，XP推荐立即建立这部分设计的可执行原型。实现并评估设计原型被称为spike解决方案。其目的是在真正实现开始时就降低风险，对可能存在设计问题的故事确认其最初的估计。

XP鼓励的重构既是构建技术又是设计技术，Fowler[Fow00]描述的重构如下：

重构是以不改变代码外部行为而改进其内部结构的方式来修改软件系统的过程。这是一种净化代码（并修改或简化内部设计）以尽可能减少引入错误的严格方法。实质上，重构就是在编码完成之后改进代码设计。

因为XP设计实际上不使用符号并且几乎不产生工作产品，如果有的话也只不过是生成除CRC卡和spike解决方案之外的工作产品，所以设计

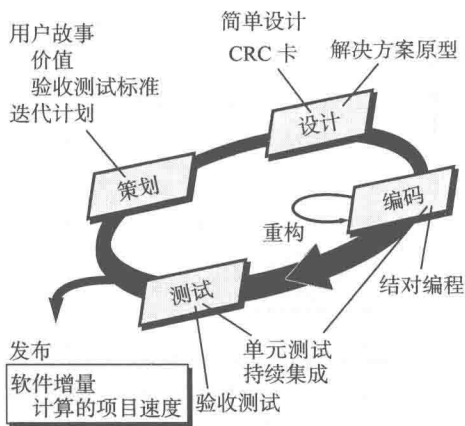


图 5-2 极限编程过程

关键点 项目速度是团队生产力的精妙度量。

建议 XP并不强调设计的重要性。这一点并不是所有人都认同。事实上，有些时候，应当强调设计。

网络资源 重构技术及其工具可在网站 www.refactoring.com 找到。

关键点 重构是以不改变其外部功能或行为而改进设计（或源代码）的内部结构。

① 虽然复杂的设计表示和术语可以加以简化，但每种软件工程方法都应遵循这些设计指导。

② 面向对象的类将在本书附录2、第9章中讨论，并贯穿本书的第二部分。

会被当作是可以并且应当在构建过程中连续修改的临时人工制品。重构的目的是控制那些“可以根本改进设计”的小的设计变更所要进行的修改[Fow00]。然而应当注意的是,重构所需的工作量随着应用软件规模的增长而急剧增长。

XP的中心观念是设计可以在编码开始前后同时进行,重构意味着设计随着系统的构建而连续进行。实际上,构建活动本身将给XP团队提供关于如何改进设计的指导。

编码。XP推荐在故事开发和初步设计完成之后,团队不是直接开始编码,而是开发一系列用于检测本次(软件增量)^①发布的包括所有故事的单元测试,一旦建立起单元测试^②,开发者就更能够集中精力于必须实现的内容以通过单元测试。不需要加任何额外的东西(KIS,保持简洁)。一旦编码完成,就可以立即完成单元测试,从而向开发者提供即时反馈。

编码活动中的关键概念(也是XP中被讨论得最多的方面之一)是结对编程。XP建议两个人面对同一台计算机共同为一个故事开发代码。这一方案提供了实时解决问题(两个人总比一个人强)和实时质量保证的机制(在代码写出后及时得到复审),同时也使得开发者能集中精力于手头的问题。实施过程中,不同成员担任的角色略有不同,例如,一名成员考虑设计特定部分的编码细节,而另一名成员确保编码遵循特定的标准(XP所要求的那些),或者确保故事相关的代码满足已开发的单元测试,并根据故事进行验证。^③

当结对的两人完成其工作后,他们所开发代码将与其他人的工作集成起来。有些情况下,这种集成作为集成团队的日常工作实施。还有一些情况下,结对者自己负责集成,这种“连续集成”策略有助于避免兼容性和接口问题,建立能及早发现错误的“冒烟测试”环境(第17章)。

测试。所建立的单元测试应当使用一个可以自动实施的框架(因此易于执行并可重复),这种方式支持每当代码修改(会经常发生,为XP提供重构理论)之后即时的回归测试策略(第17章)。

一旦将个人的单元测试组织到一个“通用测试集”[Wel99],便每天都可以进行系统的集成和确认测试。这可以为XP团队提供连续的进展指示,也可在一旦发生问题的时候及早提出预警。Wells[Wel99]指出,“每几个小时修改一些小问题,要比仅仅在最后截止期之前修正大问题要节省时间。”

XP验收测试也称为客户测试,由客户规定技术条件,并且着眼于客户可见的、可评审的系统级的特性和功能,验收测试根据本次软件发布中所实现的用户故事而确定。

网络资源 XP
的有用信息见于
www.xprogr ammi ng.com。

提问 什么是结对编程?

建议 许多软件团队都是由个人构成的。必须努力改变这种文化,应该说结对编程很奏效。

提问 单元测试如何在XP中使用?

关键点 XP验收测试是由用户故事驱动的。

5.4.2 工业极限编程

Joshua Kerievsky[Ker05]这样描述工业极限编程(IXP):“IXP是XP的一种有机进化。它由XP的最低限要求、以客户为中心和测试驱动精神组成。IXP与原来XP的主要差别在

① 这种方法类似于开始学习之前先了解考题。通过只集中注意力在将要回答的问题上,这使得研究要容易得多。

② 第17章详细地讨论了单元测试,集中于单个软件构件,检查构件接口、数据结构及功能,其目的在于发现构件内的错误。

③ 结对编程已在整个软件界普及,关于该主题,《华尔街日报》[Wal12]用头版进行了报道。

于其管理具有更大的包容性，它扩大了用户角色，升级了技术实践。”IXP 合并了六个新实践，这些新实践的设计是为了有助于确保在大型组织内的一些重要项目中，XP 工作能够成功地实施。

准备评估。IXP 团队确定该项目社区的所有成员（例如利益相关者、开发者、管理者）是否都准备就绪，是否建立了合适的环境，以及是否理解所涉及的技术水平。

项目社区。IXP 团队确定人员及其所具有的技能是否合适，是否针对该项目已进行了阶段性培训。该“社区”包括技术专家和其他利益相关者。

项目特许。IXP 团队通过对项目本身进行评估来确定对于项目的合适的商业调整是否存在，以及是否可以进一步深化组织机构的整体目标和目的。

测试驱动管理。IXP 团队建立一系列可测量的“目标”[Ker05]，以评估迄今为止的进展情况，然后定义一些机制来确定是否已经实现了这些目标。

回顾。IXP 团队在一个软件增量交付之后要实施特定的技术评审。这种评审称为回顾，评审通过软件增量或者全部软件的发布过程复查“问题、事件以及经验教训”[Ker05]。

持续学习。鼓励（可以激励）XP 团队的成员去学习新的方法和技术，从而获得高质量的软件产品。

除了以上讨论的六个新实践，IXP 还修改了大量已有的 XP 实践，重新定义了特定的角色和职责，使他们更适合大型组织的重大项目。对于 IXP 的进一步讨论，请访问网站 <http://industrialxp.org>。

提问 为了生成 IXP，在 XP 上附加了哪些新的实践？

引述 能力是你能够做什么，激励决定了你做什么，而态度决定了你做得怎样。

Lou Holtz

SafeHome 考虑敏捷软件开发

[场景] Doug Miller 的办公室。

[人物] Doug Miller，软件工程经理；Jamie Lazor 和 Vinod Raman，软件团队成员。

[对话]

（敲门，Jamie 和 Vinod 来到 Doug 的办公室。）

Jamie: Doug，有时间吗？

Doug: 当然，Jamie，什么事？

Jamie: 我们考虑过昨天讨论的过程了……就是我们打算为这个新的 SafeHome 项目选什么过程。

Doug: 哦？

Vinod: 我和在其他公司的一位朋友聊，他告诉了我极限编程。那是一种敏捷过程模型，听说过吗？

Doug: 听说过，有好也有坏。

Jamie: 对，看起来很适合我们。可以使软件开发更快，用结对编程来达到实时质量检查……我想这一定很酷。

Doug: 它确实有很多实实在在的好主意。比如，我喜欢其中的结对编程概念，还有利益相关者参加项目组的想法。

Jamie: 哦？你是说市场部将和项目组一起工作？

Doug (点头): 他们也是利益相关者，不是吗？

Jamie: 哇，他们会每隔 5 分钟就提出一些变更。

Vinod: 不要紧。我的朋友说 XP 项目有包容变更的方法。

Doug: 所以你俩认为我们应当使用 XP？

Jamie: 绝对值得考虑。

Doug: 我同意。既然我们选择了增量模

型方法，那就没有理由不利用 XP 带来的好处。

Vinod：Doug，刚才你说“有好处也有坏处”，坏处是什么？

Doug：我不喜欢 XP 不重视分析和设计……简而言之就是直接编码。（团队成员相视而笑。）

Doug：那你们同意用 XP 方法吗？

Jamie（代表二人说）：老板，我们干的就

是编码！

Doug（大笑）：没错，但我希望看到你花少量时间编码和重新编码，而花多一点时间分析我们应当做什么并设计一个可用系统。

Vinod：或许我们可以二者兼用，带有一定纪律性的敏捷。

Doug：我想我们能行，Vinod，实际上我坚信这样的做法没问题。

5.5 其他敏捷过程模型

软件工程的历史是由散乱着的几十个废弃的过程描述和方法学、建模方法和表示法、工具以及技术所构成，每一个都是轰轰烈烈地冒出来，接着又被新的且（期望是）更好的所替代。随着敏捷过程模型的大范围推广（每一种模型都在争取得到软件开发界的认可），敏捷运动正在遵循着同样的历史步伐^①。

就像前一节提到的，在所有敏捷过程模型中使用最广泛的就是 XP。但是提出的许多其他敏捷过程模型也在整个行业中使用。在本节中，我们简要概述了四种常见的敏捷方法：Scrum、DSSD、敏捷建模（AM）以及敏捷统一过程（AUP）。

5.5.1 Scrum

Scrum（得名于橄榄球比赛^②）是 Jeff Sutherland 和他的团队在 20 世纪 90 年代早期发展的一种敏捷过程模型，近年来，Schwaber 和 Beedle 对其做了进一步的发展 [Sch01b]。

Scrum 原则与敏捷宣言是一致的，应用 Scrum 原则指导过程中的开发活动，过程由“需求、分析、设计、演化和交付”等框架性活动组成。每一个框架活动中，发生于一个过程模式（在以下段落中讨论）中的工作任务称为一个冲刺（sprint）。冲刺中进行的工作（每一个框架活动中冲刺的数目根据产品复杂度和规模大小而有所不同）适应于当前的问题，由 Scrum 团队规定并常常进行实时修改。Scrum 过程的全局流程如图 5-3 所示。

Scrum 强调使用一组“软件过程模式”[Noy02]，这些过程模式被证实在时间紧张的需求变更的和业务关键的项目中是有效的。每一个过程模式定义一系列开发活动。

引述 我们这个专业实施方法论就如同 14 岁的少年穿衣服一样，还不成熟。

Stephen Hawrysh,
Jim Ruprecht

网络资源 关于 Scrum 的有用信息和资源见于 www.controlch aos.com。

① 这并不是坏事。在某个模型或方法被当作事实上的标准之前，它都在尽力争取软件工程师群体的人心。最终胜利者将发展成为最佳实践，而失败者将销声匿迹或是被融入取胜的模型。

② 一组球员围着球排列成圆圈，共同努力（有时具有暴力性）地想要带球扑地。

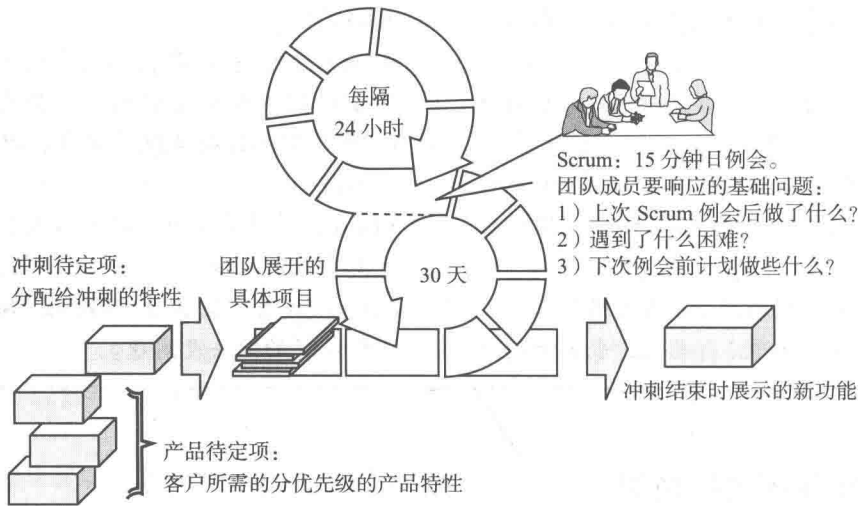


图 5-3 Scrum 过程流

待定项 (backlog) —— 一个能为用户提供商业价值的项目需求或特性的优先级列表。待定项中可以随时加入新项 (这就是变更的引入)。产品经理根据需要评估待定项并修改优先级。

冲刺 (sprint) —— 由一些工作单元组成, 这些工作单元是达到待定项中定义的需求所必需的, 并且必须能在预定的时间段 (time-box[⊖]) 内 (一般情况下为 30 天) 完成。冲刺过程中不允许有变更 (例如积压工作项)。因此, 冲刺给开发团队成员的工作提供了短期但稳定的环境。

Scrum 例会——Scrum 团队每天召开的短会 (一般情况为 15 分钟), 会上所有成员要回答三个问题 [Noy02]:

- 上次例会后做了什么?
- 遇到了什么困难?
- 下次例会前计划做些什么?

团队领导 (也称为 Scrum 主持人) 主持会议并评价每个团队成员的表现。Scrum 会议帮助团队尽早发现潜在的问题。同时, 每日例会能够促进“知识社会化交流” [Bee99] 以及自我组织团队的建设。

演示——向客户交付软件增量, 为客户演示所实现的功能并由客户对其进行评价。需提醒的很重要一点是, 演示不需要包含计划内的所有功能, 但是规定该时间段内的可交付功能必须完成。

Beedle 和他的同事 [Bee99] 在他们所发表的关于这些模式的综合讨论中提到: “Scrum 预先假定混乱的存在……”。Scrum 过程模式保证软件开发团队在无法消除不确定的世界里能成功地工作。

5.5.2 动态系统开发方法

动态系统开发方法 (Dynamic System Development Method, DSDM) [Sta97] 是一种敏

关键点 Scrum 由含有以下内容的一组过程模式构成: 强调项目优先次序、分离的工作单元、沟通以及频繁的客户反馈。

⊖ Time-box 是一个项目管理术语 (见本书第四部分), 指的是分配给完成某一任务的时间段。

捷软件开发方法,该方法提供一种框架,使其“通过在可控项目环境中使用增量原型开发模式以完全满足对时间有约束的系统的构建和维护”[CCS02]。DSDM 建议借用修改版 Pareto (佩瑞多)原则的哲学观念。这种情况下,如果交付整个应用系统需用 100% 时间,那么 80% 的应用系统可以用 20% 的时间交付。

DSDM 使用迭代软件过程,每一个迭代都遵循 80% 原则,即每个增量只完成能够保证顺利进入下一增量的工作,剩余的细节则可以在知道更多业务需求或者提出并同意变更之后完成。

DSDM 协会 (www.dsdm.org) 是一个由一些共同充当 DSDM 方法管理者的成员公司所组成的全球性组织。协会定义了一个称为 DSDM 生命周期的敏捷过程模型。该生命周期以建立基本的业务需求和约束的可行性研究开始,之后是识别功能和信息需求的业务研究。DSDM 定义了以下 3 个不同的迭代周期。

功能模型迭代——为客户开发一系列可证明其功能的增量原型(注意:所有 DSDM 原型都倾向于逐渐演化为可交付的应用系统)。这一迭代周期的意图是在用户使用原型系统时引导出反馈信息以获取补充的需求。

设计和构建迭代——在功能模型迭代中,重新构建原型以确保每一个原型都以工程化方式实现,并能为最终用户提供可操作的业务价值。有些情况下,功能模型迭代、设计和构建迭代可同步进行。

实现——将最终软件增量(一个可操作的原型)置于运行环境中。应当注意:(1) 增量不见得 100% 完成;(2) 增量置于运行环境以后可能需要变更。在这两种情况下,DSDM 开发转向功能模型迭代活动继续进行。

DSDM 和 XP (5.4 节) 可以结合使用,这种组合方法用具体实践 (XP) 定义固定的过程模型 (DSDM 生命周期),这些具体实践是构建软件增量所必需的。

5.5.3 敏捷建模

很多情况下,软件工程师必须构建大型的、业务关键型的系统,这种系统的范围和复杂性必须通过建模方式保证以下事项:(1) 所有参与者可以更好地理解要做什么;(2) 有效地将问题分解给要解决它的人;(3) 对正在设计和构建的系统质量进行评估。但在某些情况下,管理所需的符号量可能是艰巨的,艰巨性在于所建议采用模型形式化的程度、用于大型项目时模型的大小和变更发生时维护的难度。软件工程建模的敏捷方法能否为解决这些问题提供可选方案呢?

在敏捷建模官方网站上,Scott Ambler[Amb02a]用以下方式描述敏捷建模 (Agile Modeling, AM):

AM 是一种基于实践的方法学,用于对基于软件的系统实施有效建模和文档编制。在软件开发项目中,AM 是可以有效并以轻量级方式用于软件建模的标准、原则和实践。由于敏捷模型只是大致完善,而不要求完美,因此敏捷模型比传统的模型更有效。

AM 采纳了与敏捷宣言一致的全部标准。敏捷建模的指导思想认为,敏捷团队必须有做出决定的勇气,哪怕这些决定可能否决当前的设计并导致重新构建。敏捷团队也必须保持谦逊作风,应当意识到技术不能解决所有问题,要虚心尊重并采纳业务专家和其他利益相关者

网络资源 有关 DSDM 的有用资源见 www.dsdm.org。

关键点 DSDM 是过程框架,可以采用另一种敏捷方法的策略,如 XP。

网络资源 有关敏捷建模更为全面的信息见于 www.agilemodeling.com。

的意见。

虽然 AM 提出一系列的“核心”和“补充”建模原则，但其中独具特色的是 [Amb02a] 以下原则。

有目的模型。在构建模型之前，使用 AM 的开发者心中应当有明确的目标（如与客户沟通信息，或有助于更好地理解软件的某些方面），一旦确定模型的目标，则该用哪种类型的表达方式以及所需要的具体细节程度都是显而易见的。

使用多个模型。描述软件可以使用多种不同的模型和表示法，大多数项目只用到其中很小的部分就够了。AM 建议从需要的角度看，每一种模型应当表达系统的不同侧面，并且应使用能够为预期读者提供价值的那些模型。

轻装上阵。随着软件工程工作的进展，只保留那些能提供长期有价值的模型，抛弃其余的模型。保留下来的每一个工作产品都必须随着变更而进行维护，这些描述工作将使整个团队进度变慢。Ambler[Amb02a] 提示说，每次决定保留一个模型，你都要在以抽象方式使用信息的便利性与敏捷性方面做权衡（即团队内部、团队与利益相关者增强沟通）。

内容重于表述形式。建模应当向预期的读者分享重要的信息。一个有用内容很少但语法完美的模型不如一个带有缺陷但能向读者提供有用内容的模型更有价值。

理解模型及工具。理解每一个模型及其构建工具的优缺点。

适应本地需要。建模方法应该适应敏捷团队的需要。

当前软件工程领域大都已采用了统一建模语言（UML）^①作为首选的方法来表示分析和设计模型。统一过程（第4章）已经为 UML 应用提供了一个框架。Scott Ambler[Amb06] 已经开发出集成了其敏捷建模原理的 UP 的简单版本。

5.5.4 敏捷统一过程

敏捷统一过程（Agile Unified Process, AUP）采用了一个“在大型上连续”以及“在小型上迭代”[Amb06] 的原理来构建基于计算机的系统。采用经典 UP 阶段性活动——起始、细化、构建和转换——AUP 提供了一系列活动（例如软件工程活动的一个线性序列），能够使团队为软件项目构想出一个全面的过程流。然而，在每一个活动里，一个团队迭代使用敏捷，并且将有意义的软件增量尽可能快地交付给最终用户。每个 AUP 迭代执行以下活动 [Amb06]。

- **建模。**建立对商业和问题域的 UML 表述。然而，为了保持敏捷，这些模型应当“足够好”[Amb06]，以使团队继续前进。
- **实现。**将模型翻译成源代码。
- **测试。**像 XP 一样，团队设计和执行一系列的测试来发现错误以保证源代码满足需求。
- **部署。**就像第3章中讨论过的一般过程活动，这部分的部署重点仍然是对软件增量的交付以及获取最终用户的反馈信息。

引述 几天前我去药店打算买些感冒药，可是不容易啊。看到有太多的药可选，往前走，是一种速效的，还有一种是长效的。究竟选哪个好呢？是图眼前还是图长远呢？

Jerry Seinfeld

建议 对所有软件工程工作而言，“轻装”都是适合的指导思想。我们只需要那些有价值的模型，不要多，也不要少。

① 在本书附录1中提供了UML的引论。

- 配置及项目管理。在 AUP 中，配置管理（第 21 章）着眼于变更管理、风险管理以及对开发团队的任一常效产品^①的控制。项目管理追踪和控制开发团队的工作进展并协调团队活动。
- 环境管理。环境管理协调过程基础设施，包括标准、工具以及适用于开发团队的支持技术。

虽然 AUP 与统一建模语言有历史上和技术上的关联，但是很重要的一点必须注意，UML 模型可以与本章所讲的任一敏捷过程模型相结合。

软件工具 敏捷开发

[目标] 敏捷开发工具的目标是辅助敏捷开发的一个或多个方面，强调便利地快速构建可执行软件。这些工具也可以用于惯例（传统）过程模型（第 4 章）的开发。

[机制] 工具的机制各不相同。通常，敏捷工具集包括项目计划、用例开发和需求收集、快速设计、代码生成和测试的自动支持。

[代表性工具]^②

注意：由于敏捷开发是一个热门话题，因此大多数软件工具供应商都声称出售支持敏捷方法的工具。下面的工具具有的特

性对敏捷项目非常有用。

- OnTime。由 Axosoft 开发（www.axosoft.com），提供对各种技术活动的敏捷过程管理支持。
- Ideogramic UML。由 Ideogramic 开发（<http://ideogramic-uml.software.informer.com>），是特别为敏捷过程开发的 UML 工具集。
- Together Tool Set。由 Borland 销售（www.borland.com），提供支持 XP 和其他敏捷过程中许多技术活动的工具包。

5.6 敏捷过程工具集

敏捷哲学的拥护者指出，自动软件工具（例如设计工具）应当被看作是对开发团队活动小小的补充，而不是团队成功的关键。然而，Alistair Cockburn[Coc04] 建议，工具是有益处的，“敏捷团队强调使用工具可以达到快速理解的目的。有些工具是社会性的，甚至开始于租赁阶段。有些工具是技术性的，可以帮助使用者团队模拟物理现状。很多工具是物理性的，允许人们在工作场所操作这些工具。”

协作和沟通“工具”通常技术含量较低，并且与可以提供信息以及协调敏捷开发人员的任何机制相结合（“这些机制可以是物理上的近距离性、白板、海报、索引卡以及可粘贴的留言条”[Coc04] 或现代社会网络技术）。积极的沟通是通过团队能动性获得的（例如结对编程），而被动的沟通是通过“信息辐射体”实现的（例如，一台平面显示器可以显示一个增

关键点 这里所提到的敏捷过程“工具集”更多地是从人员方面而不是从技术方面支持敏捷过程。

① 常效工作产品指的是被管理的团队在不同阶段开发的模型、文档或测试用例，这些产品在软件增量交付后并不废弃。

② 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

量不同组件的全部状态)。项目管理工具降低了甘特 (Gantt) 图的重要性, 人们使用挣值图 (earned value chart) 来取代甘特图或“所创建的测试与所通过的测试对比图……还有其他工具可用来优化敏捷团队工作的环境 (例如更有效的会议区), 通过培养社交互动 (例如配置团队)、物理设备 (例如电子白色书写板)、过程增强 (例如结对编程或时间段)” [Coc04] 等提高团队文化。

所有这些都是工具吗? 如果它们能够促进敏捷团队成员的工作并提高最终产品的质量, 那么它们就是工具。

习题与思考题

- 5.1 重新阅读本章开头的“敏捷软件开发宣言”, 你能否想出一种情况, 此时四个权值中的一个或多个将使软件开发团队陷入麻烦?
- 5.2 用自己的话描述 (用于软件项目的) 敏捷性。
- 5.3 为什么迭代过程更容易管理变更? 是不是本章所讨论的每一个敏捷过程都是迭代的? 只用一次迭代就能完成项目的敏捷过程是否存在? 解释你的答案。
- 5.4 是否每一个敏捷过程都可以用第 3 章所提及的通用框架活动来描述? 建一张表, 将通用活动和每个敏捷过程所定义的活动对应起来。
- 5.5 试着再加上一条“敏捷性原则”, 以便帮助软件工程团队更具有机动性。
- 5.6 选择 5.3.1 节提到的一条敏捷性原则, 讨论本章所描述的各过程模型是否符合该原则。(注意: 我们只是给出了这些过程模型的概述, 因此, 确定一个或多个模型是否符合某个原则或许不可能, 需要做更多的研究 (对这个问题并不需要)。)
- 5.7 为什么需求变更这么大? 人们终究无法确定他们想要什么吗?
- 5.8 大多数敏捷过程模型推荐面对面交流。然而, 现在软件开发团队成员及其客户在地理上是相互分散的。你是否认为应该避免这种地理上的分散? 能否想出一个办法克服这个问题?
- 5.9 写出一个 XP 用户故事, 描述适用于大部分网页浏览器的“最喜欢的地方”或“最喜欢的事物”特性。
- 5.10 什么是 XP 的 spike 解决方案?
- 5.11 用自己的话描述 XP 的重构和结对编程的概念。
- 5.12 利用第 3 章介绍的过程模式模板, 为 5.5.1 节提出的任意一个 Scrum 模式开发出过程模式。
- 5.13 访问敏捷建模官方网站, 列出所有核心的和补充性的 AM 原则。
- 5.14 5.6 节中给出的工具集为敏捷方法的“软”方面提供了很多支持。由于交流非常重要, 因此请推荐一种实际工具集, 用来增强敏捷团队中利益相关者之间的交流。

扩展阅读与信息资源

敏捷软件开发的全部理论和基本原则在本章提供的很多参考书中都有深入的论述。另外, 在 Pichler (《Agile Project Management with Scrum: Creating Products that Customers Love》, Addison-Wesley, 2010)、Highsmith (《Agile Project Management: Creating Innovative Products》, 2nd ed. Addison-Wesley, 2009)、Shore 和 Chromatic (《The Art of Agile Development》, O'Reilly Media, 2008)、Hunt (《Agile Software Construction》, Springer, 2005) 以及 Carmichael 和 Haywood (《Better Software Faster》, Prentice Hall, 2002) 的书中给出了关于该主题的有效讨论。Aguanno (《Managing Agile Projects》, Multi-media Publications, 2005) 和 Larman (《Agile and Iterative Development: A Manager's Guide》, Addison-Wesley, 2003) 介绍了关于管理的概述及项目管理问题的思考。Highsmith (《Agile Software

Development Ecosystems》, Addison-Wesley, 2002) 介绍了关于敏捷原则、过程和实践的调查。Booch 及其同事 (《Balancing Agility and Discipline》, Addison-Wesley, 2004) 给出了关于平衡敏捷性和规范性的颇有价值的讨论。

Martin (《Clean Code: A Handbook of Agile Software Craftsmanship》, Prentice-Hall, 2009) 指出了在敏捷软件工程环境下开发“干净代码”所需的敏捷原则、模式和实践。Leffingwell (《Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise》, Addison-Wesley, 2011) 和《Scaling Software Agility: Best Practices for Large Enterprises》, Addison-Wesley, 2007) 讨论了在大型项目中扩大敏捷实践的策略。Lippert 和 Rook (《Refactoring in Large Software Projects: Performing Complex Restructurings Successfully》, Wiley, 2006) 讨论了在大型、复杂系统中重构的使用。Stamelos 和 Sftesos (《Agile Software Development Quality Assurance》, IGI Global, 2007) 讨论了符合敏捷理论的 SQA 技术。

在过去的 10 年间, 已经有很多书描述了极限编程。Beck (《Extreme Programming Explained: Embrace Change》, 2nd ed., Addison-Wesley, 2004) 的书依然是本主题的权威著作。另外, Jeffries 及其同事 (《Extreme Programming Installed》, Addison-Wesley, 2000)、Succi 和 Marchesi (《Extreme Programming Examined》, Addison-Wesley, 2001)、Newkirk 和 Martin (《Extreme Programming in Practice》, Addison-Wesley, 2001) 以及 Auer 及其同事 (《Extreme Programming Applied: Play to Win》, Addison-Wesley, 2001) 的著作中, 就如何更好地应用 XP 给出了关键的有指导意义的讨论。McBreen (《Questioning Extreme Programming》, Addison-Wesley, 2003) 则以挑剔的眼光审视 XP, 定义了其何时、何地更为适用。对结对编程的深入考虑可阅读 McBreen (《Pair Programming Illuminated》, Addison-Wesley, 2003)。

Kohut (《Professional Agile Development Process: Real World Development Using SCRUM》, Wrox, 2013)、Rubin (《Essential Scrum: A Practical Guide to the Most Popular Agile Process》, Addison-Wesley, 2012)、Larman 和 Vodde (《Scaling Lean and Agile Development: Thinking and Organizational Tools for Large Scale Scrum》, Addison-Wesley, 2008) 以及 Schwaber (《The Enterprise and Scrum》, Microsoft Press, 2007) 讨论了对主营业务产生影响的项目中使用 Scrum。Cohn (《Succeeding with Agile》, Addison-Wesley, 2009) 以及 Schwaber 和 Beedle (《Agile Software Development with SCRUM》, Prentice-Hall, 2001) 发表了对 Scrum 方法的深入探讨。DSDM Consortium (《DSDM: Business Focused Development》, 2nd ed., Pearson Education, 2003) 和 Stapleton (《DSDM: The Method in Practice》, Addison-Wesley, 1997) 的书提供了关于 DSDM 方法的有价值的论述。

Ambler 和 Lines (《Disciplined Agile Delivery: A Practitioner's Guide to Agile Delivery in the Enterprise》, IBM Press, 2012) 以及 Poppendieck (《Lean Development: An Agile Toolkit for Software Development Managers》, Addison-Wesley, 2003) 为管理和控制敏捷工程提供了指导。Ambler 和 Jeffries (《Agile Modeling》, Wiley, 2002) 深入探讨了敏捷建模。

可以从网上获得关于敏捷软件开发的更多信息资源, 有关敏捷过程的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。

软件工程的人员方面

要点浏览

概念：我们都在努力学习最新的编程语言、最好的新型设计方法、最流行的敏捷过程或最新发布的热门软件工具。但说到底，是人在开发计算机软件。因此，在软件工程中，人对一个项目的成功所起的作用和那些最新最好的技术是一样的。

人员：软件工程工作是由个人和团队完成的。在某些情况下，个人要承担大部分的责任，但大多数行业级软件要由团队完成。

重要性：只有团队的活力恰到好处，软件团队才能成功。软件工程师有时会给人不好相处的印象。但实际上，在要构建的产品中，团队里的工程师与同事及其他利益相关者进行良好合作是非常必要的。

步骤：首先，你要了解并不断积累一个成功的软件工程师应具备的个人特质。然后，你需要提升软件工程中应具备的综合心理素质，这样才能在进行项目时少走弯路，避免失误。而且，你要理解软件团队的结构和动态，因为基于团队的软件工程在行业中很常见。最后，你需要重视社交媒体、云端和其他协作工具的影响力。

工作产品：对人员、过程和最终产品有更好的洞察力。

质量保证措施：花时间去观察成功的软件工程师是如何工作的，并调整自己的方法来利用他们所展现的优点。

在《IEEE 软件》的一期特刊中，客邀编辑们 [deS09] 做出如下评论：

软件工程包括大量可以改善软件开发过程和最终产品的技术、工具和方法。技术不断进步并产生了很多鼓舞人心的成果。然而，软件不单是用适合的技术方案解决不适合的技术手段而创造出的一种产品。软件由人开发、被人使用并支持人与人之间的互动。因此，人的特质、行为和合作是实际的软件开发的重心。

在本章之后的章节中，我们将讨论构建成功的软件产品所需的“技术、工具和方法”。但在此之前，我们有必要知道，如果没有技术娴熟并积极参与的人员，那么软件项目是不太可能成功的。

6.1 软件工程师的特质

你想成为软件工程师吗？很显然，你需要掌握技术，学习并运用那些理解问题所需的技能，设计有效的解决方案，构造软件并努力测试以达到质量最优。你需要管理变更，与利益相关者沟通，并在合适的情况下使用合适的工具。这些我们都会在本书后面章节中用大量篇幅进行讨论。

关键概念

- 敏捷团队
- 云计算
- 合作开发环境 (CDE)
- 全球化团队
- 有凝聚力的团队
- 心理学
- 角色
- 社交媒体
- 团队属性
- 团队结构
- 团队毒性
- 特性
- XP 团队

但有些事和上述的这些同样重要——就是人的方面，掌握这些方面会让你成为卓有成效的软件工程师。Erdogmus[Erd09]指出软件工程师个人要想展现“非常专业的”行为，应具备七种特质。

一个卓有成效的软件工程师有个人责任感。这会让软件工程师去努力实现对同伴、其他利益相关者和管理者的承诺。为获得成功的结果，他会在需要的时候不遗余力地做他需要做的事情。

一个卓有成效的软件工程师对一些人的需求有敏锐的意识，这些人包括同团队的成员、对现存问题的软件解决方法有需求的利益相关者、掌控整个项目并能找到解决方法的管理者。他会观察人们工作的环境，并根据环境和人本身来调整自己的行为。

一个卓有成效的软件工程师是坦诚的。如果发现了有缺陷的设计，他会用诚实且有建设性的方式指出错误。即使被要求歪曲与进度、特点、性能、其他产品或项目特性有关的事实，他也会选择实事求是。

一个卓有成效的软件工程师会展现抗压能力。前面我们提到，软件工程工作经常处在混乱的边缘。压力（以及由此产生的混乱）来自很多方面——需求和优先级的变更、要求苛刻的利益相关者或同伴、不切实际或者令人难以忍受的管理者。但一个卓有成效的软件工程师可以在不影响业绩的情况下很好地处理这些压力。

一个卓有成效的软件工程师有高度的公平感。他会乐于与同事分享荣誉，努力避免利益冲突并且绝不破坏他人劳动成果。

一个卓有成效的软件工程师注重细节。这并不意味着追求完美，而是说他会利用产品和项目已有的概括性标准（如性能、成本、质量），在日常工作基础上仔细思考，进而做出技术性决策。

最后，一个卓有成效的软件工程师是务实的。他知道软件工程不是要恪守教条的宗教信仰，而是可根据当下情景需要进行调整的科学规则。

引述 大多数优秀的程序员不是因为报酬或公众关注而做编程，而是因为兴趣。
Linus Torvalds

提问 一个高效率的软件工程师需要具备哪些个人特质？

6.2 软件工程心理学

在一篇有关软件工程心理学的学术论文中，Bill Curtis 和 Diane Walz[Cur90] 针对软件开发提出了一种分层的行为模式（图 6-1）。在个人层面，软件工程心理学注重待解决的问题、解决问题所需的技能以及在模型外层建立的限制内解决问题的动机。在团队和项目层面，团队能动性成为主要因素。在这一层面，成功是由团队结构和社会因素决定的。团队交流、合作和协调同单个团队成员的技能同等重要。在外部层面，有组织的行为控制着公司的行为及其对商业环境的应对方式。

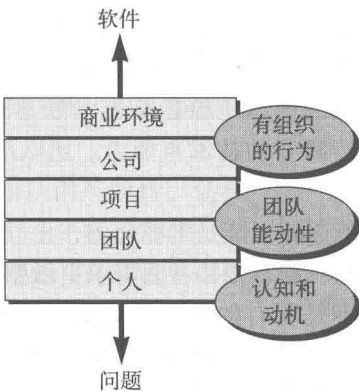


图 6-1 软件工程中的行为模式层（改自 [Cur90]）

在团队层面，Sawyer 和他的同事 [Saw08] 认为团队经常会制造虚拟的界线，这种界线会限制交流，因而会降低团队效率。他们提出了一组“跨界角色”，这能让软件团队成员在不同团队之间有效推进工作。下面这些角色要么是被特定指派的，要么是在工作中自然而然衍生出来的。

- 外联员——代表团队就时间和资源问题与外部的顾客谈判，并取得利益相关者的反馈。
- 侦察员——突破团队界线收集组织信息。“侦察活动包括：审视外部市场，寻求新技术，确定团队外界的相关活动，弄清潜在对手的活动。” [Saw08]
- 守护员——保护团队工作产品和其他信息产品。
- 安检员——把控利益相关者和他人向团队传送的信息。
- 协调员——注重横跨团队及组织内部的交流（如与组织内部的专家团队讨论特定的设计问题）。

提问 软件团队中的成员都会担任哪些角色？

6.3 软件团队

Tom DeMarco 和 Tim Lister[DeM98] 在他们的经典著作《Peopleware》中讨论了软件团队的凝聚力：

在商业领域内，我们经常使用团队这个词，把任何一组被派到一起工作的人称为一个“团队”。但是很多这样的组合并不像是团队。这些组合中的人对成功没有共同的认识，或者没有明确的团队精神。他们缺少的就是凝聚力。

提问 什么是具有凝聚力的团队？

一个有凝聚力的团队中的人应该能强烈认识到整体比个体的简单相加更强大。

一旦团队凝聚起来，成功的可能性就会变大。整个团队会变得势不可当、坚不可摧……他们不需要传统的管理方式，当然也不需要外界的推动。他们本来就有动力。

DeMarco 和 Lister 认为有凝聚力的团队中的成员比其他人员生产力更高也更有动力。他们有共同的目标、共同的文化，而且在大多数情况下，一种“精英意识”会让他们变得独特。

现在还没有能打造具有凝聚力团队的万无一失的方法。但高效的软件团队总是会显现一些特征^①。Miguel Carrasco[Car08] 认为高效的团队必须建立目标意识。比如，如果说团队成员认同团队的目标是开发可以转化产品类别的软件，并为此让公司一跃成为行业的领跑者，就可以说他们有很强的目标意识。高效的团队还必须有参与意识，让每个成员都能感受到自己的技能得到了发挥，所做出的贡献是有价值的。

关键点 高效的软件团队是多样化的，是由具备目标意识、参与意识、信任意识和进步意识的人组成的。

高效的团队应该能培养信任意识。团队中的软件工程师应该相信同伴和其管理者的技术与能力。团队应该鼓励进步意识，定期审视软件工程方法并寻求改善途径。

最高效的团队是多样化的，由具备不同技能的人员组成。技术高超的工程师会与技术背景较弱但对利益相关者的需求更敏感的队员搭档。

但不是所有的团队都高效，也不是所有的团队都具有凝聚力。事实上，很多团队都在遭受着 Jackman[Jac98] 所说的“团队毒性”。她定义了五个“可能形成有害团队环境”的因素：（1）混乱的工作氛围；（2）会造成团队成员分裂的挫败；（3）“支离破碎或协调不当”的软件过程；（4）对软件团队中角色

提问 为什么团队会没有凝聚力？

① Bruce Tuckman 发现成功的团队在取得成果的工程中会经历四个阶段（形成，争执，规范，执行）(<http://www.realsoftwaredevelopment.com/7-key-attributes-of-high-performance-software-development-teams/>)。

的模糊定义；(5)“持续且重复性的失败”。

为避免混乱的工作环境，团队应获得工作所需的所有信息。一旦确定了主要目标，不到必要时刻就不轻易改变。为避免挫败，软件团队应该尽可能地对所做的决定负责。通过了解要完成的产品、完成工作的人员以及允许团队来选择过程模型，可以避免不当的软件过程（如不必要或过于繁重的任务，或者选择错误的工作产品）。团队自身要建立责任机制（技术评审是完成这一事项的好方法），在团队成员出现失误时找出正确的方法。最后，避免陷入失败氛围的关键是建立以团队为基础的反馈和问题解决技巧。

除了 Jackman 提到的五个毒性，软件团队还经常遇到团队成员特质不同的问题。有些团队成员性格外向，有些性格内向。有的成员会直观地收集信息，从各种事实中提取概括性观点。而有的成员会有序地处理信息，从已知数据中搜集和整理详尽的细节。有的成员在进行逻辑性、有序的讨论之后做决定。而有的凭直觉，更愿意凭“感觉”做决定。有的成员需要组织任务的详细进度安排，以使它们能够完成项目的某些部分。而有的团队希望有更加自发性的环境，也可以存在未定论的问题。有的成员会在里程碑日期之前很久就把工作完成，以免到时候有压力，而有的人会从最后时刻的紧迫感中受到激励。本节总结了对人的差异的认识以及其他指导规则，这些有助于更好地构建具有凝聚力的团队。

引述 不是每个组合都是一个团队，不是每个团队都是有效率的。
Glenn Parker

6.4 团队结构

“最佳”团队结构取决于组织的管理风格、团队组成人员的数量以及他们的技术水平，还有整体的问题难度。Mantei[Man81]提出了一些在策划软件工程团队结构时应考虑的项目因素：(1)需解决问题的难度；(2)基于代码行或功能点[⊖]的结果程序的“规模”；(3)团队成员合作的时间（团队寿命）；(4)问题可模块化的程度；(5)所建系统的质量和可靠性；(6)交付日期要求的严格程度；(7)项目所需的社会化（交流）程度。

Constantine[Con93]针对软件工程团队提出了四种“组织模式”：

1. 封闭模式组成的团队遵循传统的权力层级模式。这样的团队在建立与之前的成果十分相似的软件时能做得很好，但以封闭模式工作时创新性上相对较弱。
2. 随机模式组成的团队是松散的，并依靠团队成员的个人自发性。在需要创新和技术性突破时，这类团队可以做得很优秀。但是很难完成“有秩序的操作”。
3. 开放模式尝试组成一种团队，既具有封闭模式团队的可控性，还具有随机模式团队创新性。成员们合作完成工作，并有丰富的交流和达成共识的决定，这些都是开放模式团队的特点。开放模式团队适合解决复杂的问题，但没有其他团队的效率高。
4. 同步模式组成的团队有赖于问题的自然区分，不需要很多的交流就可以将成员组织起来共同解决问题。

作为历史前鉴，最早的软件团队之一是被称为主程序员团队的封闭模式团队。这种结构最早由 Harlan Mills 提出，由 Baker[Bak72]建立。这类团队的核心包括：

提问 选择软件团队的结构时应考虑哪些因素？

提问 定义软件团队的结构时有哪些选择？

引述 如果想逐渐变好，就要具有竞争力；如果想指数级变好，就要合作。
作者不详

[⊖] 代码行 (Lines of Code, LOC) 和功能点可测量电脑程序规模，见第 24 章。

一个高级工程师（主程序员），负责计划、协调并审查团队的所有技术活动；技术人员（通常 2~5 人），进行分析和开发活动；一名支持工程师，协助高级工程师的工作，为最小化项目持续性的损失，有时可代替高级工程师。主程序员可以得到以下人员的服务支持：一位或者多位专家（如通信专家、数据库设计者）、支持人员（如技术写作人员、文书人员）和一个软件管理员。

对于主程序员团队结构，Constantine 的随机模式 [Con93] 观点认为，对于具有创造独立性的团队，其工作方法最好是创新的无序。尽管完成软件工作需要自由意识，但是软件工程组织的核心目标必须是将创新活力运用于创建高效能团队。

SafeHome 团队结构

[场景] Doug Miller 的办公室，优先致力于 SafeHome 软件项目。

[人物] Doug Miller（SafeHome 软件工程团队的管理者），Vinod Raman、Jamie Lazar 以及其他产品软件工程团队的成员。

[对话]

Doug：你们有机会看一下市场部准备的有关 SafeHome 的基础信息。

Vinod（点了点头，看着他的队友们）：是的，但是我们遇到了很多问题。

Doug：我们先不谈问题，我想探讨一下怎样构建团队，谁对什么问题负责……

Jamie：Doug，我对敏捷理念很感兴趣。我

觉得我们可以成为一个自发组织的团队。

Vinod：同意。鉴于时间有限，不确定因素很多，而且我们都很有能力（笑），这似乎是很好的方法。

Doug：我没意见，而且你们也知道怎么做。

Jamie（微笑并像在背诵什么一样说）：我们只做策略上的决定，谁在什么时候做什么，但是我们的任务是按时推出产品。

Vinod：还要保证质量。

Doug：很对。但是记住有一些限制。市场限制了需要制造出来的软件增量——当然是在与我们商讨的基础上。

Jamie：然后呢？

6.5 敏捷团队

在过去的 10 年里，敏捷软件开发（第 5 章）被认为放大了问题，扰乱了软件项目工作。回顾一下，敏捷理念支持：客户满意且尽早的软件增量发布，小型的充满动力的项目团队，非正式方法，最少的软件工程项目产品以及整体开发的简化。

6.5.1 通用敏捷团队

小型的并充满动力的项目团队也可称为敏捷团队，他们具备前面章节中所讨论的成功软件项目团队的很多特征（我们会在之后的章节谈到这些特征），并且能够避免产生问题的很多毒素。但是，敏捷理念强调个人（团队成员）通过团队合作可以加倍的能力，这是团队成功的关键因素。Cockburn 和 Highsmith[Coc01a] 在他们的文章中谈道：

如果一个项目中的人员都足够优秀，那么他们可以借助任意过程来完成任务。如果他们不够优秀，那么也就没有什么过程能弥补他们的不足——即“人员胜过过程”。然而，缺乏用户和决策支持也会扼杀一个项目——即“政策胜过人员”。如果不能得到足够的支持，优

关键点 敏捷团队是一个能自主计划和做出技术性决定的自发组织团队。

秀的人员也无法完成工作。

为了有效利用每个团队成员的能力,并完成项目工程过程中的高效合作,敏捷团队都是自组织的。一个自组织的团队不必保持单一的团队结构,而是综合运用6.2节中讨论的Constantine提出的随机、开放和同步模式。

很多敏捷过程模型(如Scrum)给予敏捷团队重要的自主性,允许团队自主做出完成工作必需的项目管理和技术决定。计划被保持在最低程度,团队可以选择自己的方式(如过程、方法、工具),仅受商业要求和组织标准的限制。在项目进行过程中,团队自发地及时将重点放在有益于项目特定点的个人能力上。为完成这项工作,敏捷团队可能每天都开例会,协商和同步当天必须要完成的事情。

基于例会期间获得的信息,团队将调整其方法以完成工作增量。随着时间的积累,持续的自发组织和合作可以促使团队完成软件增量工作。

6.5.2 XP 团队

作为极限编程(eXtreme Programming, XP)的一部分,Beck[Bec04a]定义了五项价值作为实施所有工作的基础——沟通、简单、反馈、勇气、尊重。任何一项价值都可以促成特定的XP活动、动作和任务。

为使敏捷团队和其他利益相关者达到有效的沟通(如建立所需的软件特性和功能),XP强调客户和开发者之间密切而非正式(口头)的合作,构建用作交流媒介的有效的隐喻^①,以便交流重要概念、获得持续反馈并避免冗长的文档。

为了达到简单,敏捷团队在设计时只是考虑当下需要而非长远需求。其目的是创建简单的设计,可以容易地用代码实现。如果必须要改进设计,以后可以对代码进行重构^②。

反馈来源于三种途径:所实现的软件本身、客户以及其他软件团队成员。通过设计和实行有效的测试策略(第17~19章),软件(通过测试结果)可以为敏捷团队提供反馈。团队可以利用单元测试作为其初始测试手段。每个类开发完成后,团队会根据其特定的功能开发单元测试来不断完善每个操作。向客户交付增量时,要用增量所实现的用户故事或者用例(第8章)进行验收测试。软件实现用例的输出、功能和行为的程度是一种形式的反馈。最终,在迭代计划中,新的需求会不断出现,团队可以就成本和进度的影响给客户快速提供快速的反馈。

Beck[Bec04a]认为严格地坚持特定XP实践是需要勇气的。换个更好的说法是要有原则。比如说,设计经常会面临要考虑长远需求的巨大压力。大多数软件团队都妥协了,并辩称“为明天做设计”可以在长期发展中节省时间和精力。XP团队必须有原则(勇气),要为今天做设计,要认识到长远需求可能会发生意想不到的变更,因此会带来设计和实现代码的大量返工。

要遵循每一项价值,XP团队极力主张其团队成员之间、其他利益相关者和团队成员之

引述 集体所有体现了一种观点:工作成果是属于整个(敏捷)团队而非组成团队的个人的。

Jim Highsmith

建议 尽量保持简单,但需认识到持续的“重构”能获得可观的时间和资源。

引述 怎样在不费力的情况下构建优秀的软件?XP就是答案。

未名

① 在XP中,隐喻是“任何人——客户、程序员和管理者——都能讲述系统如何工作的一个故事。”[Bec04a]。

② 重构是软件工程师在不改变设计(或源代码)的外部功能或行为的情况下改进其内部结构。实质上,重构可以用来改善设计或实现代码的效率、可读性或性能。

间以及间接地对软件本身的尊重。在成功发布软件增量时，团队对 XP 过程的重视就会油然而生。

6.6 社交媒体的影响

邮件、短信或者视频会议在软件工程工作过程中无处不在。但是这些交流机制不过是现代化替代品或面对面沟通机制的补充。社交媒体是多种多样的。

Begel[Beg10] 和他的同事在文章中讲到软件工程中社交媒体的发展和应用：

围绕软件开发的社会化过程……极大程度上取决于工程师的能力——找到有相似目标和互补技能的个人并将他们连接起来，使团队成员的交流和整个团队都表现得更加和谐，让他们在整个软件生命周期中进行合作和协调，并保证他们的产品在市场上能获得成功。

从某种意义上来说，这种“连接”和面对面的交流一样重要。团队规模越大，社交媒体的价值越大，如果团队在地域上是分离的，这种价值就更大了。

首先，社交网络是为软件项目设定的。软件团队可以通过网络从团队成员、利益相关者、技术人员、专家和其他商业人士那里收集经验，这些人已经受邀参与该网络（如果该网络是私有的）或任何兴趣团体（如果该网络是公有的）。而且无论出现什么状况、疑惑或难题，社交网络都可以做到这一点。社交媒体有很多不同的形式，每一种在软件工程工作中都有一席之地。

博客可用来发表一系列短文以描述系统的重要方面，或者用来发表针对尚处于开发中的软件特性或功能的看法。其重要性还可体现在“软件公司经常用博客非常有效地与他们的雇主及客户分享技术信息和观点，内外兼顾。” [Beg10]

引述 内容很重要，对话同样重要。

John Munsell

微博（如 Twitter）由软件工程网络成员使用，对关注他们的人发表短消息。因为这些文章是即时的，而且在各种移动平台上都能阅读，所以信息的分散达到了实时性。软件团队遇到问题时可以随时召开临时会议，出现难题时可以随时寻求专业帮助，也可以实时向利益相关者传达项目的某些方面。

在 targeted on-line 论坛上，参与者可以发布问题、观点、案例或任何其他相关信息。一个技术性的问题在发布之后的几分钟之内就可能得到多个“答复”。

社交网站（如 Facebook、LinkedIn）使软件开发者和相关技术人员达到分离化的连接。同一个社交网站上的“好友”可以找到具有相关应用领域或者要解决问题的知识或经验的好友的好友。以社交网络泛型为基础建立的专业性私有网络可以在组织内部使用。

大多数社交媒体都能组织有相似兴趣的用户形成“社区”。例如，一个由擅长实时嵌入式系统的软件工程师组成的社区，可以为相关领域的个人或团队之间的联系提供有效的方式，以改善他们的工作。随着社区的发展，参与者会讨论技术趋势、应用场景、新型工具和其他软件工程知识。最后，网址收藏夹网站（比如 Delicious、Stumble、CiteULike）为软件工程师或软件团队提供平台，让他们可以为有类似想法的个人组成的社交媒体社区推荐网页资源。

需要着重强调的是，在软件工程工作中使用社交媒体时，不能忽视隐私和安全问题。软件工程师所做的大多数工作可能是他们的雇主专有的，因此将其公开是有害的。所以，一定要在社交媒体的优点和私有信息不受控制的公开之间做出权衡。

6.7 软件工程中云的应用

云计算为我们提供了一种机制，以获取各种软件工作产品、人工制品以及与项目相关的信息。它在各处都能运行，并能消除很多软件项目对于设备依赖的限制；可以让软件团队成员进行独立于平台的、低风险的新型软件工具的实验，并提供对这些工具的反馈；可以提供新的分配方法和 β 软件测试；可以提供针对内容和配置管理的更先进的方法（第 21 章）。

鉴于云计算可以完成上述工作，它很有可能影响软件工程师们组织团队的方式、工作的方法、交流和连接的方式，以及管理软件项目的方式。无论团队成员使用何种平台、处于什么位置，都可以即时获取某个团队成员开发出的软件工程信息。

从根本上说，信息迅速扩散并极度扩展，这也改变了软件工程动态，并对软件工程中人的作用产生了深远的影响。

但是软件工程中的云计算不是没有风险的 [The13]。云分布在多个服务器，其构造和服务往往不受软件团队的控制。因此，云有很多缺点，且存在可靠性和安全性风险。云提供的服务越多，相对的开发环境就越复杂。这些服务是否能与其他供应商提供的服务相匹配？这体现了云服务在协同性上的风险。最后，如果云成为开发环境，其服务必须强调可用性和性能。而这些属性有时会与安全性、保密性和可靠性相冲突。

但是从人文的角度来看，与存在的风险相比，云为软件工程师提供了更多的好处。Dana Gardner [Gar09] 将其优点总结如下（带有警告）：

软件开发中任何涉及社交或合作的地方都很可能会用到云。项目管理、进度安排、任务列表、需求和缺陷管理作为核心团队功能会很好地进行自我调整，其中，沟通对于项目同步以及使团队所有成员——无论他们在哪里——处于同一场景非常重要。当然，需要严重警惕的是——如果你的公司是想设计产品中的嵌入式软件，那么不推荐使用云。想象一下得到苹果公司关于下一版 iPhone 的项目计划会怎样。

如 Gardner 所叙述的那样，云的关键优势之一是其增强了“软件开发的社会和协作方面”。在下一节中，你会更多地了解协作工具。

6.8 协作工具

Fillipo Lanubile 及其同事 [Lan10] 认为 20 个世纪的软件开发环境（SDE）已变成协作开发环境（Collaborative Development Environments, CDE）。^①他们是这样论述的：

工具对于团队成员之间的协作是很有必要的，它们能实现简易化、自动化以及对整个开发过程的控制。全球化软件工程特别需要充足的工具支持，因为距离因素对交流的消极影响直接或间接地加重了协作和控制问题。

在 CDE 中用到的很多工具和本书第二、三、四部分提到的辅助软件工程活动的工具没有什么不同。但是，有价值的 CDE 会提供为改善协同工作特别设计的一系列服务 [Fok10]。这些服务包括：

- 命名空间使项目团队可以用加强安全性和保密性的方式存储工作产品和其他信息，仅允许有权限的人访问。

引述 他们不再称其为网络，而是称其为云计算。我不纠结名称，随便叫什么都可以。

Larry Ellison

建议 云是软件工程信息的强大知识库，但你必须要考虑第 21 章所讨论的变更控制问题。

^① 协作开发环境（CDE）的概念是由 Grady Booch [Boo02] 提出的。

- 进度表可以协调会议和其他项目事件。
- 模板可以使团队成员在创造工作产品时保持一致的外形和结构。
- 度量支持可以量化每个成员的贡献。
- 交流分析会跟踪整个团队的交流并分离出模式，应用于需要解决的问题或状况。
- 工件收集可以通过回答以下问题的方式组织工作产品和其他项目制品：“是什么导致了某个特定的变更？可以跟哪个讨论过特定制品的人商讨有关变更？（团队）成员的个人工作是如何影响他人的工作的？” [Fok10]

提问 协作开发环境中通用的服务有哪些？

软件工具 协作开发环境

[目标] 随着软件开发不断全球化，软件团队需要的不仅仅是开发工具。他们需要的是一套服务，使得团队成员在本地和远程都能协作。

[机制] 此类工具和服务能使团队建立起协作工作的机制。CDE 可以实现 6.6 节所描述的多种或所有服务，同时也能为实现过程管理（第 4 章）提供本书中讨论的传统软件工程工具。

[代表性工具][⊖]

- GForge。一种包括项目和代码管理设施的协作环境 (<http://gforge.com/gf/>)。
- OneDesk。为开发者和利益相关者提供创造和管理项目工作空间的协作环境 (www.onedesk.com)。
- Rational Team Concert。一个深度的、协作生命周期管理系统 (<http://www-01.ibm.com/software/rational/products/rtc/>)。

6.9 全球化团队

在软件领域，全球化不仅仅意味着货物和服务的跨国交流。在过去的几十年中，由位于不同国家的软件团队共同建造的主要软件产品越来越多。这些全球化软件开发（GSD）团队具备传统软件团队（6.4 节）所具有的很多特点，但是 GSD 团队还会面临其他特有的挑战，包括协调、合作、交流及专业决策。本章前面已经讨论了协调、合作和交流的方法。对所有软件团队来说，决策问题因以下四个因素而变得复杂 [Gar10]：

- 问题的复杂性。
- 与决策相关的不确定性和风险。
- 结果不确定法则（比如，工作相关的决策会对另外的项目目标产生意外的影响）。
- 对问题的不同看法才是导致不同结论的关键。

对于 GSD 团队，协调、合作和沟通方面的挑战对决策具有深远的影响。图 6-2 解释了 GSD 团队所面临的距离挑战的影响。距离使交流复杂化，但同时也强调对协调的需求。距离也产生了由文化差异导致的障碍和复杂性。障碍和复杂性会减弱交流（比如信噪比的降低）。在这种动态环境中固有的问题会导致项目变得不稳定。

引述 越来越多的公司管理者在应对不同的文化。公司变得全球一体化，但团队却被分开并散布在全球。

Carlos Ghosn,
Nissan

⊖ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

虽然没有能够彻底改正图 6-2 中各种关系的银弹，但运用有效的 CDE（6.6 节）可有助于减弱距离的影响。

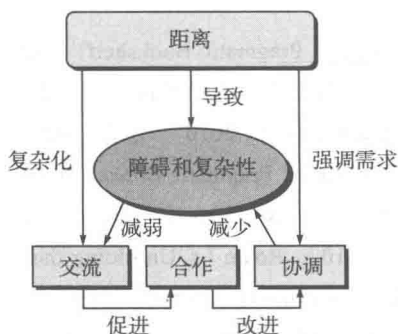


图 6-2 影响 GSD 团队的因素（改自 [Cas06]）

习题与思考题

- 6.1 根据你对优秀软件开发者的观察，说出三个共同的个人特质。
- 6.2 怎样能做到“毫无保留地诚实”，同时又不被（其他人）视为有侮辱意图或有攻击性？
- 6.3 软件团队是如何构建“人工边界”来降低与其他人交流的能力的？
- 6.4 编写一个简要场景，描述 6.2 节中的每个“跨界角色”。
- 6.5 在 6.3 节中，我们提到目标意识、参与意识、信任意识和进步意识是高效软件团队的重要属性。在团队成立时，谁应该对建立这些属性负责？
- 6.6 对于团队的四种组织模式（6.4 节）：（1）保险公司的 IT 部门；（2）国防项目承包商的软件工程小组；（3）开发电脑游戏的软件小组；（4）大型软件公司。你认为哪种最有效？并阐明理由。
- 6.7 如果要选择敏捷团队区别于传统软件团队的一个属性，你会选哪个？
- 6.8 针对 6.6 节描述的软件工程工作社交媒体的形式，选出你认为最有效的并说明理由。
- 6.9 编写场景，使 SafeHome 团队成员可以在软件项目中利用一种或多种形式的社交媒体。
- 6.10 近来，云成为计算领域中的一个热门概念。运用有关为改善软件工作而特别设计的服务的具体例子，描述云是如何提升软件组织价值的。
- 6.11 研究一下 6.8 节软件工具介绍中提到的某种 CDE 工具（或导师指定的一种工具），准备在班级中对工具的功能进行简要介绍。
- 6.12 参见图 6-2，距离为什么会使交流复杂化？距离为什么会强调对协调的需求？距离会导致哪些种类的障碍和复杂性？

扩展阅读与信息资源

很多书都讲到了软件工程中人的因素，但有两本书是公认的经典著作。Jerry Weinberg（《The Psychology of Computer Programming》，Silver Anniversary Edition, Dorset House, 1998）首次提到构建计算机软件人员的心理学。Tom DeMarco 和 Tim Lister（《Peopleware: Productive Projects and Teams》，2nd ed., Dorset House, 1999）讨论了软件开发的主要挑战在于人而非技术。

以下书籍提供了针对软件工程中人的有用观察：Mantle 和 Lichty（《Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams》，Addison-Wesley, 2012），Fowler（《The Passionate Programmer》，Pragmatic Bookshelf, 2009），McConnell（《Code Complete》，2nd ed., Microsoft Press, 2004），Brooks（《The Mythical Man-Month》，2nd ed., Addison-Wesley,

1999), Hunt 和 Tomas (《The Pragmatic Programmer》, Addison-Wesley, 1999)。Tomayko 和 Hazzan (《Human Aspects of Software Engineering》, Charles River Media, 2004) 以 XP 为重点, 讲述了软件工程中的心理学和社会学。

Rasmussen (《The Agile Samurai》, Pragmatic Bookshelf, 2010) 和 Davies (《Agile Coaching》, Pragmatic Bookshelf, 2010) 论述了敏捷开发中人的因素。有关敏捷团队的重要方面可参见 Adkins 的书 (《Coaching Agile Teams》, Addison-Wesley, 2010), 还有 Derby、Larsen 和 Schwaber 的书 (《Agile Retrospectives: Making Good Teams Great》, Pragmatic Bookshelf, 2006)。

解决问题是一种独特的人类活动, 这方面的著作有: Adair (《Decision Making and Problem Solving Strategies》, Kogan Page, 2010), Roam (《Unfolding the Napkin》, Portfolio Trade, 2009), Wananabe (《Problem Solving 101》, Portfolio Hardcover, 2009)。

Tabaka (《Collaboration Explained》, Addison-Wesley, 2006) 提供了如何在软件团队中进行合作的指导。Rosen (《The Culture of Collaboration》, Red Ape Publishing, 2009)、Hansen (《Collaboration》, Harvard Business School Press, 2009) 和 Sawyer (《Group Genius: The Creative Power of Collaboration》, Basic Books, 2007) 提供了改善技术团队合作的策略和实践性指导。

以培养人员创新性为主题的书有: Gray、Brown 和 Macanuso (《Game Storming》, O'Reilly Media, 2010), Duggan (《Strategic Intuition》, Columbia University Press, 2007) 和 Hohmann (《Innovation Games》, Addison-Wesley, 2006)。

Ebert (《Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing》, Wiley-IEEE Computer Society Press, 2011) 描述了全球软件开发的整体情况。Mite 及其同事 (《Agility Across Time and Space: Implementing Agile Methods in Global Software Projects》, Springer, 2010) 编写了有关全球开发中使用敏捷团队的论文集。

网上有讨论软件工程中人员方面的大量信息资源, 软件过程相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

建模

在本书这一部分，读者将学到构建高质量需求模型和设计模型的基本原则、概念和方法。接下来的章节将涉及如下问题：

- 什么是需求工程？什么是能导致良好需求分析的基本概念？
- 如何创建需求模型？它包含哪些元素？
- 好的设计包含哪些元素？
- 体系结构设计如何为其他的设计活动建立架构，使用什么模型？
- 如何设计高质量软件部件？
- 在设计用户接口时使用什么概念、模型和方法？

一旦解决了以上问题，准备工作也就完成了，可以马上开始软件工程应用实践了。

理解需求

要点浏览

概念: 在开始任何技术工作之前, 关注于一系列需求工程任务都是个好主意。这些任务有助于你理解软件将如何影响业务、客户想要什么以及最终用户将如何和软件交互。

人员: 软件工程师 (在 IT 界有时指系统工程师或分析员) 以及项目的其他利益相关者 (项目经理、客户、最终用户) 都将参与需求工程。

重要性: 在设计和开发某个优秀的计算机软件时, 如果软件解决的问题是错误的, 那么即使软件再精巧也满足不了任何人的要求。这就是在设计和开发一个基于计算机的系统之前理解客户需求的重要性所在。

步骤: 需求工程首先是起始阶段 (定义将要解决的问题的范围和性质); 然后是获

取 (帮助利益相关者定义需要什么); 接下来是细化 (精确定义和修改基本需求)。当利益相关者提出问题后, 就要进行协商 (如何定义优先次序? 什么是必需的? 何时需要?) 最后, 以某种形式明确说明问题, 再经过评审或确认以保证我们和利益相关者对于问题的理解是一致的。

工作产品: 需求工程的目的在于为各方提供关于问题的一份书面理解。不过依然可以得到工作产品: 用户场景, 功能和特性列表, 需求模型或规格说明。

质量保证措施: 利益相关者评审需求工程的工作产品, 以确保你所理解的正是他们真正想要的。需要提醒大家的是: 即使参与各方均认可, 事情也会有变化, 而且变化可能贯穿整个项目实施过程。

理解问题的需求是软件工程师所面对的最困难的任务之一。第一次考虑这个问题时, 开发一个清晰且易于理解的需求看起来似乎并不困难。毕竟, 客户难道不知道需要什么? 最终用户难道对将给他们带来实际收益的特性和功能没有清楚的认识? 不可思议的是, 很多情况下的确是这样的。甚至即使用户和最终用户清楚地知道他们的要求, 这些要求也会在项目的实施过程中改变。

在 Ralph Young[You01] 的一本关于有效需求实践的书的前言中, 我写道:

这是最恐怖的噩梦: 一个客户走进你的办公室, 坐下, 正视着你, 然后说: “我知道你认为你理解我说的是, 但你并不理解的是, 我所说的并不是我想要的。” 这种情况总是在项目后期出现, 而当时的情况通常是: 已经做出交付期限的承诺, 声誉悬于一线并且已经投入大量资金。

关键概念

- 分析模式
- 协作
- 细化
- 获取
- 起始
- 协商
- 质量功能部署
- 需求工程
- 需求收集
- 需求管理
- 需求监控
- 规格说明
- 利益相关者

我们这些已经在系统和软件业中工作多年的人就生活这样的噩梦中，而且目前还都不知道该怎么摆脱。我们努力从客户那里获取需求，但却难以理解获取的信息。我们通常采用混乱的方式记录需求，而且总是花费极少的时间检验到底记录了什么。我们容忍变更控制自己，而不是建立机制去控制变更。总之，我们未能为系统或软件奠定坚实的基础。这些问题中的每一个都是极富挑战性的，当这些问题集中在一起时，即使是最有经验的管理人员和实际工作人员也会感到头痛。但是，确实存在解决方法。

把需求工程称作以上所述挑战的“解决方案”可能并不合理，但需求工程确实为我们提供了解决这些挑战的可靠途径。

7.1 需求工程

设计和编写软件富有挑战性、创造性和趣味性。事实上，编写软件是如此吸引人，以至于很多软件开发人员在清楚地了解用户需要什么之前就迫切地投入到编写工作中。开发人员认为：在编写的过程中事情总是会变得清晰；只有在检验了软件的早期版本后项目利益相关者才能够更好地理解要求；事情变化太快，以至于理解需求细节是在浪费时间；最终要做的是开发一个可运行的程序，其他都是次要的。构成这些论点的原因在于其中也包含了部分真实情况^①，但是这中间的每个论点都存在一些小问题，汇集在一起就可能导致软件项目的失败。

需求工程（Requirement Engineering, RE）是指致力于不断理解需求的大量任务和技术。从软件过程的角度来看，需求工程是一个软件工程动作，开始于沟通并持续到建模活动。它必须适用于过程、项目、产品和人员的需要。

需求工程在设计 and 构建之间建立起联系的桥梁。桥梁源自何处？有人可能认为源于项目利益相关者（如项目经理、客户、最终用户），也就是在他们那里定义业务需求、刻画用户场景、描述功能和特性、识别项目约束条件。其他人可能会建议从宽泛的系统定义开始，此时软件只是更大的系统范围中的一个构件。但是不管起始点在哪里，横跨这座桥梁都将把我们带到项目之上更高的层次：允许由软件团队检查将要进行的软件工作的内容；必须提交设计和构建的特定要求；完成指导工作顺序的优先级定义；以及将深切影响随后设计的信息、功能和行为。

在过去的几十年，有很多技术变革影响着需求工程的过程 [Wev11]。无处不在的计算使计算机技术能够与许多日常事务相结合。这些事务通过联网就能生成更多完整的用户信息，同时伴随着对隐私和安全问题的关注。

在电子市场广泛传播的应用引领了更多各式各样利益相关者的需求。利益相关者能定制产品，以满足一小部分最终用户特定的需求目标。当产品开发周期缩短时，流水线型需求工程会有压力，目的是推动产品更快进入市场。但是存在

关键概念

利益相关者
用例
确认
观点
工作产品

引述 构建一个

软件系统最困难的部分是确定构建什么。其他工作不会像这部分工作一样，在出错之后会如此严重地影响随后实现的系统，并且在以后修补竟会如此困难。

Fred Brooks

关键点 需求工程为设计和构造奠定了坚实的基础。如果没有需求工程，那么实现的软件很有可能无法满足客户的需求。

建议 可以在需求阶段做一些设计工作，在设计阶段做一些需求工作。

^① 对小项目（不超过一个月）和只涉及简单的软件工作的更小项目，这确实是正确的。但随着软件规模和复杂性的增加，这些论点就开始出问题了。

的根本问题仍然是如何及时获得精准稳定的利益相关者的输入信息。

需求工程包括七项明确的任务：起始，获取，细化，协商，规格说明，确认和管理。注意，这些需求工作中的一些任务会并行发生，并且要全部适应项目的要求。

起始。如何开始一个软件项目？有没有一个独立的事件能够成为新的基于计算机的系统或产品的催化剂？需求会随时间的流逝而发展吗？这些问题没有确定的答案。某些情况下，一次偶然的交谈就可能导致大量的软件工程工作。但是多数项目都是在确定了商业要求或是发现了潜在的新市场、新服务时才开始。业务领域的利益相关者（如业务管理人员、市场人员、产品管理人员）定义业务用例，确定市场的宽度和深度，进行粗略的可行性分析，并确定项目范围的工作说明。所有这些信息都取决于变更，但是应该充分地[Ⓐ]与软件工程组织[Ⓐ]及时讨论。在项目起始阶段中[Ⓐ]，要建立基本的理解，包括存在的问题、谁需要解决方案、所期望解决方案的性质、与项目利益相关者和开发人员之间达成初步交流合作的效果。

获取。询问客户、用户和其他人：系统或产品的目标是什么，想要实现什么，系统和产品如何满足业务的要求，最终系统或产品如何用于日常工作。这些问题看上去是非常简单的，但实际上并非如此，而是非常困难。

获取过程中最重要的是建立商业目标 [Cle10]。我们的工作就是与利益相关者约定，鼓励他们诚实地分享目标。一旦抓住目标，就应该建立优先机制，并（为满足利益相关者的目标）建立潜在架构的合理性设计。

引述 大多数软件灾难的种子通常都是在软件项目开始的头三个月内种下的。

Caper Jones

信息栏 基于目标导向的需求工程

目标是一个系统或产品必须达到的长期目的。目标可能涉及功能性或非功能性（例如可靠性、安全性、可用性等）内容。目标通常是向利益相关者解释需求的好方法，一旦建立了目标，就可以在利益相关者中处理冲突和矛盾。

从目标中可以系统地推出目标模型（第9章和第10章）和需求。展示目标之间各种链接的目标图，能提供一定程度的追踪性，从高层次的策略关注到低层次的

技术细节。目标应该精确定义，并为需求的细化、验证与确认、冲突管理、协商、解释和发展提供服务基础。

需求检测中的冲突通常是目标自身存在冲突的结果。通过一套相互商定的目标可获得冲突解决方案，这些目标与每个成员及利益相关者的渴求相一致。有关目标和需求工程更完备的讨论可以查找 Lamsweweerde 的论文 [LaM01b]。

Christel 和 Kang[Cri92] 指出了一系列问题，可以帮助我们理解为什么获取需求这么困难。范围问题发生在系统边界不清楚的情况下，或是客户和用户的说明带有不必要的技术细节，这些细节可能会导致混淆而不是澄清系统的整体目标。理解问题发生在客户和用户并不

Ⓐ 如果要开发一个基于计算机的系统，那么讨论将从系统工程过程开始。请访问 www.mhhe.com/pressman 获取更多系统工程的讨论详情。

Ⓐ 应该记得（第4章）统一过程定义了更全面的“起始阶段”，包括本章所讨论的起始、获取和细化工作。

完全确定需要什么的情况下,包括:对其计算环境的能力和限制所知甚少,对问题域没有完整的认识,与系统工程师在沟通上存在问题,忽略了那些他们认为是“明显的”信息,确定的需求和其他客户及用户的要求相冲突,需求说明有歧义或不可测试。易变问题发生在需求随时间推移而变更的情况下。为了帮助解决这些问题,需求工程师必须以有组织的方式开展需求收集活动。

细化。在起始和获取阶段获得的信息将在细化阶段进行扩展和提炼。该任务的核心是开发一个精确的需求模型(第8~10章),用以说明软件的功能、特征和信息的各个方面。

细化是由一系列的用户场景建模和求精任务驱动的。这些用户场景描述了如何让最终用户和其他参与者与系统进行交互。解析每个用户场景以便提取分析类——最终用户可见的业务域实体。应该定义每个分析类的属性,确定每个类所需要的服务^①,确定类之间的关联和协作关系,并完成各种补充图。

协商。业务资源有限,而客户和用户却提出了过高的要求,这是常有的事。另一个相当常见的现象是,不同的客户或用户提出了相互冲突的需求,并坚持“我们的特殊要求是至关重要的”。

需求工程师必须通过协商过程来调解这些冲突。应该让客户、用户和其他利益相关者对各自的需求排序,然后按优先级讨论冲突。使用迭代的方法给需求排序,评估每项需求的成本和风险,处理内部冲突,删除、组合或修改需求,以便参与各方均能达到一定的满意度。

规格说明。在基于计算机的系统(和软件)的环境下,术语规格说明对不同的人有不同的含义。规格说明可以是一份写好的文档、一套图形化的模型、一个形式化的数学模型、一组使用场景、一个原型或上述各项的任意组合。

有人建议应该开发一个“标准模板”[Som97]并将之用于规格说明,他们认为这样将促使以一致的从而也更易于理解的方式来表示需求。然而,在开发规格说明时保持灵活性有时是必要的,对大型系统而言,文档最好采用自然语言描述和图形化模型来编写。而对于技术环节明确的较小产品或系统,使用场景可能就足够了。

提问 为什么获得对客户需要的清晰理解会非常困难?

建议 细化是件好事,但你必须知道何时可以停止细化。关键是不能采用为设计建立一个坚实基础的方式说明问题。如果超出这个点就是在做设计。

建议 在有效的协商中没有赢家也没有输家,而是双赢。这是因为双方合作才是“交易”的坚实基础。

关键点 规格说明的形式和规格随着待开发软件的规模和复杂度的不同而变化。

信息栏 软件需求规格说明模板

软件需求规格说明(SRS)是在项目商业化之前必须建立的详细描述软件各个方面的工作产品。值得注意的是,常常没有正规的SRS。事实上很多实例表明,在软件需求规格说明上投入的工作量还不如投入到其他软件工程活动中。然而,如果

软件由第三方开发,当缺少规格说明导致严重业务问题时,或是当系统非常复杂或业务十分重要时,才能表明需求规格说明是非常必要的。

Process Impact公司的Karl Wiegers [Wie03]开发了一套完整的模板(参考

① 服务通过对类的封装操作数据,也可使用术语“操作”和“方法”。如果你不熟悉面向对象的概念,附录2有基本的入门指导。

www.processimpact.com/process_assets/srs_template.doc), 能为那些必须建立完整需求规格说明书的人提供指导。主题大纲如下:

目录

版本历史

1. 引言

1.1 目的

1.2 文档约定

1.3 适用人群和阅读建议

1.4 项目范围

1.5 参考文献

2 总体描述

2.1 产品愿景

2.2 产品特性

2.3 用户类型和特征

2.4 操作环境

2.5 设计和实施约束

2.6 用户文档

2.7 假设和依赖

3. 系统特性

3.1 系统特性 1

3.2 系统特性 2(等等)

4 外部接口需求

4.1 用户接口

4.2 硬件接口

4.3 软件接口

4.4 通信接口

5. 其他非功能需求

5.1 性能需求

5.2 安全需求

5.3 保密需求

5.4 软件质量属性

6. 其他需求

附录 A: 术语表

附录 B: 分析模型

附录 C: 问题列表

每个需求规格说明主题的详细描述可以从前面所提的 URL 下载 SRS 模板来得到。

确认。在确认这一步将对需求工程的工作产品进行质量评估。需求确认要检查规格说明[⊖]以保证: 已无歧义地说明了所有的系统需求; 已检测出不一致性、疏忽和错误并予以纠正; 工作产品符合为过程、项目和产品建立的标准。

正式的技术评审是最主要的需求确认机制。确认需求的评审小组包括软件工程师、客户、用户和其他利益相关者, 他们检查系统规格说明, 查找内容或解释上的错误, 以及可能需要进一步澄清的地方、丢失的信息、不一致性(这是建造大型产品或系统时遇到的主要问题)、冲突的需求或是不可实现的(不能达到的)需求。

为了说明发生在需求验证过程中的某些问题, 要考虑两个看似无关紧要的需求:

- 软件应该对用户友好。
- 成功处理未授权数据库干扰的比率应该小于 0.0001。

第一个需求对开发者而言概念太模糊, 以至于不能测试或评估。什么是“用户友好”的精确含义? 为了确认它, 必须以某种方式使其量化。

第二个需求是一个量化元素(“小于 0.0001”), 但干扰测试会很困难且很费时。这种级别的安全真的能保证应用系统吗? 其他附加的与安全相关的需求(例如密码保护、特定的握手协议)能代替指明的定量需求吗?

建议 需求确认时的一个重要问题是 consistency。使用分析模型可以保证需求说明的一致性。

⊖ 每个项目有不同的规格说明特性。在某些情况下, 规格说明是指收集到的用户场景或其他一些事物。在另一些情况下, 规格说明可以包括用户场景、模型和说明性文档。

Glinz[Gli09] 写到质量需求要以恰当的方式表述, 从而保证交付最优价值。这意味着要对不能满足利益相关者质量要求的交付系统进行风险(第26章)评估, 并且试图以最小代价减轻风险。质量需求越关键, 越需要采用量化术语来陈述。在某些情况下, 常见质量需求可以使用定性技术进行验证(例如用户调查或检查表)。在其他情况下, 质量需求可以使用定性和定量相结合的评估方式进行验证。

信息栏 需求确认检查单

通常, 按照检查单上的一系列问题检查每项需求是非常有用的。这里列出其中部分可能会问到的问题:

- 需求说明清晰吗? 有没有可能造成误解?
- 需求的来源(如人员、规则、文档)弄清了吗? 需求的最终说明是否已经根据或对照最初来源检查过?
- 需求是否用定量的术语界定?
- 其他哪些需求和此需求相关? 是否已经使用交叉索引矩阵或其他机制清楚地加以说明?
- 需求是否违背某个系统领域的约束?
- 需求是否可测试? 如果可以, 能否说明检验需求的测试(有时称为确认准则)?
- 对已经创建的任何系统模型, 需求是否可追溯?
- 对整体系统/产品目标, 需求是否可追溯?
- 规格说明的构造方式是否有助于理解、轻松引用和翻译成更技术性的工作产品?
- 对已创建的规格说明是否建立了索引?
- 和系统性能、行为及运行特征相关的需求说明是否清楚? 哪些需求是隐含出现的?

需求管理。对于基于计算机的系统, 其需求会变更, 而且变更的要求贯穿于整个生命周期。需求管理是用于帮助项目组在项目进展中标识、控制和跟踪需求以及需求变更的一组活动^①。这类活动中的大部分和第21章中讨论的软件配置管理(SCM)技术是相同的。

软件工具 需求工程

[目标] 需求工程工具有助于需求收集、需求建模、需求管理和需求确认。

[机制] 工具的工作机制多种多样。通常, 需求工程工具创建大量的图形化(例如UML)模型用以描述系统的信息、功能和行为。这些模型构成了软件过程中其他所有活动的基础。

[代表性工具]^②

Volere 需求资源网站 www.volere.co.uk/tools.htm 提供了一个相当全面(也是最新)的需求工程工具列表。我们将在第8、9章讨论需求建模工具。下面提到的工具主要侧重于需求管理。

- EasyRM。由 Cybernetic Intelligence GmbH

① 正规的需求管理只适用于具有数百个可确认需求的大型项目。对于小项目, 该需求工程工作可以适当裁剪, 一定程度的不正规也是可以接受的。

② 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名被各自的开发者注册为商标。

开发 (<http://www.visuresolutions.com/visure-requirement-software>), 可视化需求是一套灵活完整的需求工程生命周期解决方案, 支持需求捕获、分析、规格说明、确认和验证、管理和复用。

- Rational RequisitePro。由 Rational 软件开发 (www-03.ibm.com/software/products/

us/en/reqpro/), 允许用户建立需求数据库, 表述需求之间的关系, 并且组织、排序和跟踪需求。

从前面所提的 Volere 网站和 www.jiludwig.com/requirements_management_tools.html 还可以找到很多额外的需求管理工具。

7.2 建立根基

在理想情况下, 利益相关者和软件工程师在同一个小组中工作^①。在这种情况下, 需求工程就只是和组里熟悉的同事进行有意义的交谈。但实际情况往往不是这样。

客户或最终用户可能位于不同的城市或国家, 对于想要什么可能仅有模糊的想法, 对于将要构建的系统可能存在有冲突的意见, 他们的技术知识可能很有限, 而且只有有限的时间与需求工程师沟通。这些事情都是不希望遇到的, 但却又是十分常见的, 软件开发团队经常被迫在这种环境的限制下工作。

下节将要讨论启动需求工程所必需的步骤, 以便理解软件需求, 使得项目自始至终走向成功解决方案的方向。

7.2.1 确认利益相关者

Sommerville 和 Sawyer[Som97] 把利益相关者定义为“直接或间接地从正在开发的系统中获益的人”。可以确定如下几个容易理解的利益相关者: 业务运行管理人员、产品管理人员、市场销售人员、内部和外部客户、最终用户、顾问、产品工程师、软件工程师、支持和维护工程师以及其他人员。每个利益相关者对系统都有不同的考虑, 当系统成功开发后所能获得的利益也不相同, 同样, 当系统开发失败时所面临的风险也是不同的。

在开始阶段, 需求工程师应该创建一个人员列表, 列出那些有助于获取需求的人员(7.3节)。最初的人员列表将随着接触的利益相关者人数的增多而增加, 因为每个利益相关者都将被询问“你认为我还应该和谁交谈”。

7.2.2 识别多重观点

因为存在很多不同的利益相关者, 所以系统需求调研也将从很多不同的视角开展。例如, 市场销售部门关心能激发潜在市场的、有助于新系统销售的功能和特性; 业务经理关注应该在预算内实现的产品特性, 并且这些产品特性应该满足已规定的市场限制; 最终用户可能希望系统的功能是他们所熟悉的并且易于学习和使用; 软件工程师可能关注非技术背景的利益相关者看不到的软件基础设施, 使其能够支持更多的适于销售的功能和特性; 支持工程师可能关注软件的可维护性。

建议 利益相关者是那些对将要开发的系统有直接的兴趣或直接从中获益的人。

引述 把三个利益相关者请进一个房间, 然后问他们想要什么样的系统, 你很可能得到四个或更多的不同观点。

作者不详

^① 推荐所有项目都使用该方法, 而且该方法是敏捷软件开发方法的主要部分。

这些参与者（以及其他人员）中的每一个人都将为需求工程贡献信息。当从多个角度收集信息时，所形成的需求可能存在不一致性或是相互矛盾。需求工程师的工作就是把所有利益相关者提供的信息（包括不一致或是矛盾的需求）分类，分类的方法应该便于决策制定者为系统选择一个内部一致的需求集合。

为使软件满足用户而获取需求的过程存在很多困难：项目目标不清晰，利益相关者的优先级不同，人们还没说出的假设，相关利益者解释含义的不同，很难用一种方式对陈述的需求进行验证 [Ale11]。有效需求工程的目标是去除或尽力减少这些问题的发生。

7.2.3 协同合作

假设一个项目中有五个利益相关者，那么对一套需求就会有五种或更多的正确观点。在前面几章中我们已经注意到客户（和其他利益相关者）之间应该团结协作（避免内讧），并和软件工程师团结协作，这样才能成功实现预定的系统。但是如何实现协作？

需求工程师的工作是标识公共区域（即所有利益相关者都同意的需求）和矛盾区域（或不一致区域，即某个利益相关者提出的需求和其他利益相关者的需求相矛盾）。当然，后一种矛盾区域的解决更有挑战性。

信息栏 使用“优先点”

有一个方法能够解决相互冲突的需求，同时更好地理解所有需求的相对重要性，那就是使用基于“优先点”的“投票”方案。所有的利益相关者都会分配到一定数量的优先点，这些优先点可以适用于很多需求。在需求列表上，每个利益相关者通过向每个需求分配一个或多个优先点来

表明（从他的个人观点）该需求的相对重要性。优先点用过之后就不能再次使用，一旦某个利益相关者的优先点用完，他就不能再对需求实施进一步的操作。所有利益相关者在每项需求上的优先点总数显示了该需求的综合重要性。

协作并不意味着必须由委员会定义需求。在很多情况下，利益相关者的协作是提供他们各自关于需求的观点，而一个有力的“项目领导者”（例如业务经理或高级技术员）可能要对删减哪些需求做出最终判断。

7.2.4 首次提问

在项目开始时的提问应该是“与环境无关的” [Gau89]。第一组与环境无关的问题集中于客户和其他利益相关者以及整体目标和收益。例如，需求工程师可能会问：

- 谁是这项工作的最初请求者？
- 谁将使用该解决方案？
- 成功的解决方案将带来什么样的经济收益？
- 对于这个解决方案你还需要其他资源吗？

这些问题有助于识别所有对构建软件感兴趣的利益相关者。此外，问题还确认了某个成功实现的可度量收益以及定制软件开发的可选方案。

引述 知道问题比知道所有的答案更好。

James Thurber

下一组问题有助于软件开发组更好地理解问题，并允许客户表达其对解决方案的看法：

- 如何描述由某成功的解决方案产生的“良好”输出的特征？
- 该解决方案强调解决了什么问题？
- 能向我们展示（或描述）解决方案使用的商业环境吗？
- 存在将影响解决方案的特殊的性能问题或约束吗？

最后一组问题关注沟通活动本身的效率。Gause 和 Weinberg[Gau89]称之为“元问题”。下面给出了元问题的简单列表：

- 你是回答这些问题的合适人选吗？你的回答是“正式的”吗？
- 我的提问和你想解决的问题相关吗？
- 我的问题是否太多了？
- 还有其他人员可以提供更多的信息吗？
- 还有我应该问的其他问题吗？

这些问题（和其他问题）将有助于“打破坚冰”，并有助于交流的开始，而且这样的交流对成功获取需求至关重要。但是，会议形式的问与答（Q&A）并非是一定会取得成功的好方法。事实上，Q&A 会议应该仅仅用于首次接触，然后应该用问题求解、协商和规格说明等需求获取方式来取代。在 7.3 节中将介绍这类方法。

7.3 获取需求

需求获取（又称为需求收集）将问题求解、细化、协商和规格说明等方面的元素结合在一起。为了鼓励合作，一个包括利益相关者和开发人员的团队共同完成如下任务：确认问题，为解决方案的相关元素提供建议，商讨不同的方法并描述初步的需求解决方案[Zah90]。^①

7.3.1 协作收集需求

关于需求收集，现在已经提出了很多不同的协同需求收集方法，各种方法适用于稍有不同场景，而且所有这些均是在下面的基本原则之上做了某些改动：

- 实际的或虚拟的会议由软件工程师和其他利益相关者共同举办和参与。
- 制定筹备和参与会议的规则。
- 建议拟定一个会议议程，这个议程既要足够正式，使其涵盖所有的要点，但也不能太正式，以鼓励思维的自由交流。
- 由一个“主持人”（可以是客户、开发人员或其他人）控制会议。
- 采用“方案论证手段”（可以是工作表、活动挂图、不干胶贴纸、电子公告牌、聊天室或虚拟论坛）。

协作收集需求的目标是标识问题，提出假设解决方案的相关元素，协商不同方法以及确定一套解决需求问题的初步方案。

在需求的起始阶段写下 1～2 页的“产品要求”（7.2 节），选择会议地点、时间和日期，选派主持人，邀请软件团队和其他利益相关者参加会

提问 什么问题有助于你获得对问题的初步认识？

引述 问问题的人是五分钟的傻瓜，而不问问题的人将永远是傻瓜。

中国谚语

提问 主持一个协作的需求收集会议的基本原则是什么？

网络资源 联合应用程序开发（JAD）是需求收集普遍采用的技术。可以从以下地址找到详细的说明：www.carolla.com/wp-jad.htm。

^① 这种方法有时称为协助应用规格说明技术（Facilitated Application Specification Technique, FAST）。

议。在会议日期之前，给所有参会者分发产品需求。

举一个例子^①，考虑 SafeHome 项目中的一个市场营销人员撰写的产品要求。此人对 SafeHome 项目的住宅安全功能叙述如下：

我们的研究表明，住宅管理系统市场以每年 40% 的速度增长。我们推向市场的首个 SafeHome 功能将是住宅安全功能，因为多数人都熟悉“报警系统”，所以这更容易销售。

住宅安全功能应该为防止和识别各种不希望出现的“情况”提供保护，如非法入侵、火灾、漏水、一氧化碳浓度超标等。该功能将使用无线传感器监控每种情况，户主可以用程序控制，并且在出现状况时系统将自动用电话联系监控部门。

事实上，其他人在需求收集会议中将补充大量的信息。但是，即使有了补充信息，仍有可能存在歧义性和疏漏，也有可能发生错误。但在目前的情况下，上面的“功能描述”是足够的。

在召开会议评审产品要求的前几天，要求每个与会者列出构成系统周围环境的对象、由系统产生的其他对象以及系统用来完成功能的对象。此外，要求每个与会者列出服务操作或与对象交互的服务（过程或功能）列表。最后，还要开发约束列表（如成本、规模大小、业务规则）和性能标准（如速度、精确度）。告诉与会者，这些列表不要求完备无缺，但要反映每个人对系统的理解。

SafeHome 描述的对象可能包括：一个控制面板、若干烟感器、若干门窗传感器、若干动态检测器、一个警报器、一个事件（一个已被激活的传感器）、一台显示器、一台计算机、若干电话号码、一个电话等。服务列表可能包括：配置系统、设置警报器、监控传感器、电话拨号、控制面板编程以及读显示器（注意，服务作用于对象）。采用类似的方法，每个与会者都将开发约束列表（例如，当传感器不工作时系统必须能够识别，必须是用户友好的，必须能够和标准电话线直接连接）和性能标准列表（例如，一个传感器事件应在一秒内被识别，应实施事件优先级方案）。

这些对象列表可以用一张大纸钉在房间的墙上或用便签纸贴在墙上或写在墙板上。或者，列表也可以被贴在内部网站的电子公告牌上或聊天室中，便于会议前的评审。理想情况下，应该能够分别操作每个列表项，以便于合并列表、删除项以及加入新项。在本阶段，严禁批评和争论。

当某一专题的各个列表被提出后，小组将生成一个组合列表。该组合列表将删除冗余项，并加入在讨论过程中出现的一些新的想法，但是不删除任何东西。在所有专题的组合列表都生成后，由主持人组织开始讨论。组合列表可能会缩短、加长或重新措词，以求更恰当地反映即将开发的产品或系统，其目标是为所开发系统的对象、服务、约束和性能提交一个意见一致的列表。

在很多情况下，列表所描述的对象或服务需要更多的解释。为了完成这一任务，利益相关者为列表中的条目编写小规格说明（mini-specification），或者生成包括

建议 如果一个系统或产品将要为很多用户提供服务，那么必须绝对保证需求是从所有用户的代表群中提取的。如果只有一个用户定义所有的需求，那么接受风险将会非常高。

引述 事实不会因被忽略而不存在。

Aldous Huxley

建议 一定要避免攻击客户意见“太昂贵”或“不实际”这样的严厉谴责。这里建议通过协商形成一个大家均接受的列表。为了达到这个目标，必须保持开放的思想。

① SafeHome（有一定的扩展和变动）在下面很多章节中用于说明软件工程的重要方法。作为一个练习，为该例子举行你自己的需求收集会议并为其开发一组列表是值得的。

对象或服务的用户用例（7.4节）。例如，对 SafeHome 对象控制面板的小规格说明如下：

控制面板是一个安装在墙上的装置，尺寸大概是 230mm×130mm。控制面板与传感器、计算机之间无线连接，通过一个 12 键的键盘与用户交互，通过一个 75mm×75mm 的 OLED 彩色显示器为用户提供反馈信息。软件将提供交互提示、回显以及类似的功能。

然后，将每个小规格说明提交给所有利益相关者进行讨论，进行添加、删除和进一步细化等工作。在某些情况下，编写小规格说明可能会发现新的对象、服务、约束或性能需求，可以将这些新发现加入到原始列表中。在所有的讨论过程中，团队可能会提出某些在会议中不能解决的问题，将这些问题列表保留起来以便这些意见在以后的工作中发挥作用。

SafeHome 召开需求收集会议

[场景] 一间会议室，进行首次需求收集会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人（指向白板）：这是目前住宅安全功能的对象和服务列表。

营销人员：从我们的观点看差不多覆盖了需求。

Vinod：没有人提到他们希望通过 Internet 访问所有的 SafeHome 功能吗？这应该包括住宅安全功能，不是吗？

营销人员：是的，这很正确……我们必须加上这个功能以及合适的对象。

主持人：这还需要加上一些限制吗？

Jamie：肯定，包括技术上的和法律上的。

产品代表：什么意思？

Jamie：我们必须确保外人不能非法侵入系统、使系统失效、抢劫甚至更糟。我们的责任非常重。

Doug：非常正确。

营销人员：但我们确实需要……只是保证能够制止外人进入……

Ed：说比做起来容易，而且……

主持人（打断）：我现在不想讨论这个问题。我们把它作为动作项记录下来，然后继续讨论（Doug 作为会议的记录者记下合适的内容）。

支持者：我有种感觉，这儿仍存在很多需要考虑的问题。

（小组接下来花费 20 分钟提炼并扩展住宅安全功能的细节。）

许多利益相关者关心（例如准确率、数据可用性、安全性）这些基本的非功能系统需求（7.2 节）。当利益相关者倾听这些观点时，软件工程必须考虑他们建立系统的语境。在众多问题中必须回答以下几个问题 [Lag10]：

- 我们能建立系统吗？
- 这些开发流程能让我们打败市场竞争对手吗？
- 有适当的资源可建立和维护假设的系统吗？
- 系统性能能满足客户需求吗？

随着时间的推移，这些问题或其他问题的答案都将随之发展变化。

关键点 QFD 以最大限度地满足客户的方式来定义需求。

7.3.2 质量功能部署

质量功能部署（Quality Function Deployment, QFD）是一种将客户要求转化成软件技术

需求的技术。QFD的“目的是最大限度地让客户从软件工程过程中感到满意”[Zul92]。为了达到这个目标,QFD强调理解“什么是对客户有价值的”,然后在整个工程活动中部署这些价值。

在QFD语境中,常规需求是指在会议中向客户陈述一个产品或系统时的目标,如果这些需求存在,客户就会满意。期望需求暗指在产品或系统中客户没有清晰表述的基础功能,缺少了这些将会引起客户的不满。兴奋需求是超出客户预期的需求,当这些需求存在时会令人非常满意。

虽然QFD概念可应用于整个软件工程[Par96],但是特定的QFD技术可应用于需求获取活动。QFD通过客户访谈和观察、调查以及历史数据检查(如问题报告)为需求收集活动获取原始数据。然后把这些数据翻译成需求表——称为客户意见表,并由客户和利益相关者评审。接下来使用各种图表、矩阵和评估方法抽取期望的需求并尽可能获取兴奋需求[Aka04]。

7.3.3 使用场景

收集需求时,系统功能和特性的整体愿景开始具体化。但是在软件团队弄清楚不同类型的最终用户如何使用这些功能和特性之前,很难转移到更技术化的软件工程活动中。为实现这一点,开发人员和用户可以创建一系列的场景——场景可以识别对将要构建系统的使用线索。场景通常称为用例[Jac92],它描述了人们将如何使用某一系统。在7.4节中将更详细地讨论用例。

建议 每个人都希望实现大量的“兴奋需求”,但是必须小心,那是“需求蔓延”开始的原因。然而另一方面,经常是兴奋需求才最终导致突破性的产品!

网络资源 有关QFD的更多信息请访问www.qfdi.org。

SafeHome 开发一个初步的使用场景

[场景] 一间会议室,继续首次需求收集会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins, 软件团队成员; Doug Miller, 软件工程经理; 三个市场营销人员; 一个产品工程代表; 一个会议主持人。

[对话]

主持人: 我们已经讨论过通过 Internet 访问 SafeHome 功能的安全性,我想考虑得再多些。下面我们开发一个使用住宅安全功能的用户场景。

Jamie: 怎么做?

主持人: 我们可以采用两种不同的方法完成这个工作,但是现在,我想不要太正式吧。请问(他指向一个市场人员),你设想该如何使用这样的系统?

营销人员: 嗯……好的,如果我出门在外,我想我能做的就是必须让某个没有安全码

的管家或修理工进入我家。

主持人(微笑): 这就是你的理由……告诉我们实际上你怎么做?

营销人员: 嗯……首先我需要一台电脑,然后登录为所有 SafeHome 用户提供的 Web 网站,提供我的用户账号……

Vinod(打断): Web 页面必须是安全的、加密的,以确保安全……

主持人(打断): 这是个有用信息, Vinod,但这太技术性了,我们还是只关注最终用户将如何使用该功能,好吗?

Vinod: 没问题。

营销人员: 那么,就像我所说的,我会登录一个 Web 网站并提供我的用户账号和两级密码。

Jamie: 如果我忘记密码怎么办?

主持人(打断): 好想法, Jamie。但是我们先不谈这个。我们先把这种情况作为“异

常”记录下来。一定还有其他的异常。

营销人员：在我输入密码后，屏幕将显示所有的 SafeHome 功能。我选择住宅安全功能，系统可能会要求确认我是谁，要求我的地址或电话号码或其他什么，然后显示一张图片，包括安全系统控制面板和我能执行的功能列表——安装系统、解除系

统、解除一个或多个传感器。我猜还可能允许我重新配置安全区域和其他类似的东西，但是我不确定。

（当市场营销人员继续讨论时，Doug 记录下大量内容。这些构成了最初非正式的用例场景基础。另一种方法是让市场营销人员写下场景，但这应该在会议之外进行。）

7.3.4 获取工作产品

根据将要构建的系统或产品规模的不同，需求获取后产生的工作产品也不同。对于大多数系统而言，工作产品包括：（1）要求和可行性陈述；（2）系统或产品范围的界限说明；（3）参与需求获取的客户、用户和其他相关利益者的名单；（4）系统技术环境的说明；（5）需求列表（最好按照功能加以组织）以及每个需求适用的领域限制；（6）一系列使用场景，有助于深入了解系统或产品在不同运行环境下的使用；（7）任何能够更好地定义需求的原型。所有参与需求获取的人员需要评审以上的每一个工作产品。

提问 什么样的信息是需求收集产生的？

7.3.5 敏捷需求获取

在敏捷过程中，通过向所有利益相关者询问，生成用户故事以获取需求。每个用户故事描述了一个从用户角度出发的简单系统需求。写在小记事卡片上的用户故事使开发者更容易选择和管理需求子集，以便实现下一代产品的改进。敏捷倡导者主张使用用户用自己的语言编写的记事卡片，这可以使软件开发人员的注意力转移到与利益相关者交流所选的需求，而不是他们自己的议事日程 [Mail0a]。

关键点 在敏捷过程模型中，用户故事是从客户中获取并记录需求的方式。

虽然用敏捷开发的方法进行需求获取吸引了很多软件团队，但批评人士认为这种方法常常缺少对全局商业目标和非功能需求的考量。在某些情况下，需要返工并考虑性能和安全问题。另外，用户故事可能无法为随时间发展的系统提供足够的基础。

7.3.6 面向服务的方法

面向服务的开发将系统看作一套服务的集合。服务可能“与提供单一功能一样简单，例如基于需求 / 应答机制提供一系列随机数，或者与复杂元素整合，例如 Web 服务 API” [Mic12]。

提问 在基于服务模型的语境中服务是什么？

面向服务开发的需求获取关注由应用系统所定义的服务。做个比喻，想象你进入了一家高级酒店，可享受的服务有：门童向客人问候，一位服务员替客人停车，前台服务员为客人办理入住手续，一位服务员管理着行李，客房服务员协助客人找到所安排的房间。精心设计客人与酒店员工的联系和触点会提升客人对拜访酒店时所获得服务的印象。

许多服务设计方法强调理解客户、创造性思维和快速建立解决方案 [Mail0b]，为达到这

些目标,需求获取过程要包括人类行为研究、创新工作室和早期低精度的原型^①。需求获取的技术也必须获得品牌信息和利益相关者感知的信息。另外,为了研究如何让客户选择某个品牌,分析师需要有策略地发现和记录新用户所渴求的质量需求。基于这一点,用户故事将非常有帮助。

描述触点需求的特征时,应使其与整体服务需求相呼应,因此,每种需求都应该可以追溯到一种特定的服务。

7.4 开发用例

在一本讨论如何编写有效用例的书中,Alistair Cockburn[Coc01b]写道:“一个用例捕获一份‘合同’……即说明对某个利益相关者的请求做出响应时,系统在各种条件下的行为。”本质上,用例讲述了程式化的故事:最终用户(扮演多种可能角色中的一个)如何在特定环境下和系统交互。这个故事可以是叙述性的文本、任务或交互的概要、基于模板的说明或图形表示。不管其形式如何,用例都从最终用户的角度描述了软件或系统。

撰写用例的第一步是确定故事中所包含的“参与者”。参与者是在将要说明的功能和行为环境内使用系统或产品的各类人员(或设备)。参与者代表了系统运行时人(或设备)所扮演的角色,更为正式的定义是:参与者是任何与系统或产品通信的事物,且对系统本身而言参与者是外部的。在使用系统时,每个参与者都有一个或多个目标。

要注意的是,参与者和最终用户并非一回事。典型的用户可能在使用系统时扮演了许多不同的角色,而参与者表示了一类外部实体(经常是人员,但并不总是如此),在用例中他们仅扮演一种角色。例如机床操作员(一个用户),他和生产车间(其中布置了许多机器人和数控机床)内的某个控制计算机交互。在仔细考察需求后,控制计算机软件需要4种不同的交互模式(角色):编程模式、测试模式、监控模式和故障检查模式。因此,4个参与者可定义为:程序员、测试员、监控员和故障检修员。有些情况下,机床操作员可以扮演所有这些角色,而另一些情况下,每个参与者的角色可能由不同的人员扮演。

需求获取是一个逐步演化的活动,因此在第一次迭代中并不能确认所有的参与者。在第一次迭代中有可能识别主要的参与者[Jac92],而对系统了解更多之后,才能识别出次要的参与者。主要参与者直接且经常使用软件,他们要获取所需的系统功能并从系统得到预期收益。次要参与者为系统提供支持,以便主要参与者能够完成他们的工作。

一旦确认了参与者,就可以开发用例了。对于应该由用例回答的问题,Jacobson[Jac92]提出了以下建议:^②

- 主要参与者和次要参与者分别是谁?
- 参与者的目标是什么?
- 故事开始前有什么前提条件?

关键点 从基于服务的模型中获取的需求细化了由应用场景所衍生出的服务。每一个触点都代表用户与系统的一次交互,从而获得所需的服务。

关键点 用例是从参与者的角度定义的。参与者是人员(用户)或设备在和软件交互时所扮演的角色。

网络资源 一篇非常好的关于用例的论文可以从www.ibm.com/developerworks/web_services/library/co-design7.html下载。

提问 为了开发有效的用例我需要知道什么?

① 在假设使用软件产品的环境中研究用户行为。

② Jacobson 的问题已经被扩展到为用例场景提供更复杂的视图。

- 参与者完成的主要工作或功能是什么？
- 按照故事所描述的还可能需要考虑什么异常？
- 参与者的交互中有什么可能的变化？
- 参与者将获得、产生或改变哪些系统信息？
- 参与者必须通知系统外部环境的改变吗？
- 参与者希望从系统获取什么信息？
- 参与者希望得知意料之外的变更吗？

回顾基本的 SafeHome 需求，我们定义了 4 个参与者：房主（用户）、配置管理人员（很可能就是房主，但扮演不同的角色）、传感器（附属于系统的设备）和监控子系统（监控 SafeHome 房间安全功能的中央站）。仅从该例子的目的来看，我们只考虑了房主这个参与者。房主通过使用报警控制面板或计算机等多种方式和住宅安全功能交互：（1）输入密码以便能进行其他交互；（2）查询安全区的状态；（3）查询传感器的状态；（4）在紧急情况时按下应急按钮；（5）激活或关闭安全系统。

考虑房主使用控制面板的情况，系统激活的基本用例如下。^①

1. 房主观察 SafeHome 控制面板（图 7-1），以确定系统是否已准备好接收输入。如果系统尚未就绪，“not ready”消息将显示在 LCD 显示器上，房主必须亲自动手关闭窗户或门以使得“not ready”消息消失。（not ready 消息意味着某个传感器是开着的，即某个门或窗户是开着的。）
2. 房主使用键盘键入 4 位密码，系统将该密码与已存储的有效密码相比较，如果密码不正确，控制面板将鸣叫一声并自动复位以等待再次输入，如果密码正确，控制面板将等待进一步的操作。
3. 房主选择键入“stay”或“away”（图 7-1）以启动系统。“stay”只激活外部传感器（内部的运动监控传感器是关闭的），“away”激活所有的传感器。
4. 激活时，房主可以看到一个红色的警报灯。

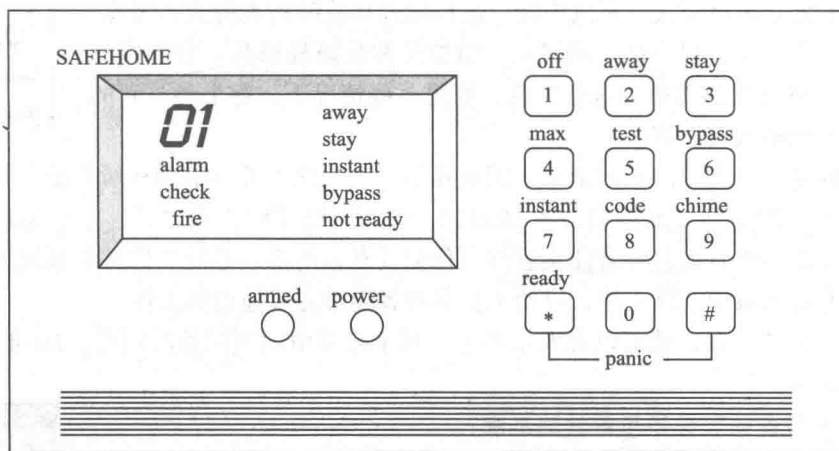


图 7-1 SafeHome 控制面板

① 注意，该用例和通过 Internet 访问系统的情形不同，该用例的环境是通过控制面板交互而不是使用计算机或移动设备所提供的图形用户接口（GUI）。

基本用例从较高层次上给出参与者和系统之间交互的故事。

在很多情况下，需要进一步细化用例以便为交互提供更详细的说明。

例如，Cockburn[Coc01b] 建议使用如下模板详细说明用例。

用例：初始化监控。

主要参与者：房主。

目标：在房主离开住宅或留在房间时，设置系统以监控传感器。

前提条件：系统支持密码输入和传感器识别功能。

触发器：房主决定“设置”系统，即打开警报功能。

场景：

1. 房主：观察控制面板。
2. 房主：输入密码。
3. 房主：选择“stay”或“away”。
4. 房主：观察红色报警灯显示 SafeHome 已经被打开。

异常：

1. 控制面板没有准备就绪：房主检查所有的传感器，确定哪些是开着的（即门窗是开着的），并将其关闭。
2. 密码不正确（控制面板鸣叫一声）：房主重新输入正确的密码。
3. 密码不识别：必须对监控和响应子系统重新设置密码。
4. 选择 stay：控制面板鸣叫两声并且 stay 灯点亮；激活边界传感器。
5. 选择 away：控制面板鸣叫三声并且 away 灯点亮；激活所有传感器。

优先级：必须实现。

何时可用：第一个增量。

使用频率：每天多次。

使用方式：通过控制面板接口。

次要参与者：技术支持人员，传感器。

次要参与者使用方式：

技术支持人员：电话线。

传感器：有线或无线接口。

未解决的问题：

1. 是否还应该有不使用密码或使用缩略密码激活系统的方式？
2. 控制面板是否还应显示附加的文字信息？
3. 房主输入密码时，从按下第一个按键开始必须在多长时间内输入密码？
4. 在系统真正激活之前有没有办法关闭系统？

可以使用类似的方法开发其他房主的交互用例。重要的是必须认真评审每个用例。如果某些交互元素模糊不清，用例评审将解决这些问题。

建议 用例通常写得不规范，但是使用这里介绍的模板可以确保你已经说明了所有关键问题。

SafeHome 开发高级用例图

[场景] 会议室，继续需求收集会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed

Robbins，软件团队成员；Doug Miller，

软件工程经理；三个市场营销人员；一个

产品工程代表；一个会议主持人。

[对话]

主持人：我们已经花费了相当多的时间讨论 SafeHome 住宅安全功能。在休息时我画了一个用例图，用它来概括重要的场景，这些场景是该功能的一部分。大家看一下。

(所有的与会者注视图 7-2。)

Jamie：我恰好刚开始学习 UML 符号^①。住宅安全功能是由中间包含若干椭圆的大方框表示吗？而且这些椭圆代表我们已经用文字写下的用例，对吗？

主持人：是的。而且棍型小人代表参与者——与系统交互的人或事物，如同用例中所描述的……哦，我使用作了标记的矩形表示在这个用例中那些不是人而是传感器的参与者。

Doug：这在 UML 中合法吗？

主持人：合法性不是问题，重点是交流信息。我认为使用棍型小人代表设备可能会产生误导，因此我做了一些改变。我认为这不会产生什么问题。

Vinod：好的。这样我们就为每个椭圆进行

了用例说明，还需要生成更详细的基于模板的说明吗？我们已经阅读过那些说明了。

主持人：有可能，但这可以等到考虑完其他的 SafeHome 功能之后。

营销人员：等一下，我已经看过这幅图，突然间我意识到我们遗漏了什么。

主持人：哦，是吗。告诉我们遗漏了什么。
(会议继续进行。)

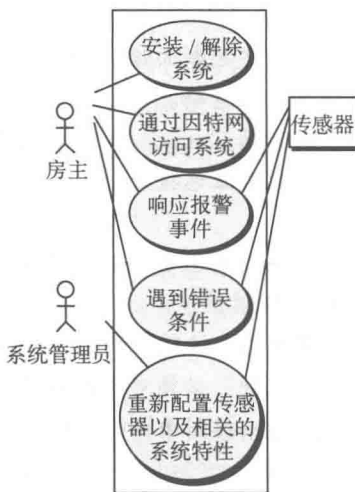


图 7-2 SafeHome 住宅安全系统功能的 UML 用例图

软件工具 | 用例开发

[目标] 通过使用能提高访问透明性和一致性的自动化模板和机制来协助开发用例。

[机制] 工具的原理各不相同。通常，用例工具为创建有效用例提供填空式的模板。大多数用例功能嵌入一系列更宽泛的需求工程功能中。

[代表性工具]^②

大量的分析建模工具（多数基于 UML）可为用例开发和建模提供文字和图形化支持。

- Objects by Design。UML 工具资源 (www.objectsbydesign.com/tools/umltools_byCompany.html)，提供对该类工具的全面链接。

7.5 构建分析模型^③

分析模型的作用是为基于计算机的系统提供必要的信息、功能和行为域的说明。随着软

① 不熟悉 UML 符号的读者请参考附录 1 中的 UML 基本指南。

② 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名被各自的开发者注册为商标。

③ 在本书中我把分析模型和需求模型作为同义词使用，它们都用于描述信息、功能和行为领域的问题需求。

件工程师更多地了解将要实现的系统以及其他相关利益者更多地了解他们到底需要什么，模型应能够动态变更。因此，分析模型是任意给定时刻的需求快照，我们对这种变更应有思想准备。

随着分析模型的演化，某些元素将变得相对稳定，为后续设计任务提供稳固的基础。但是，有些模型元素可能是不稳定的，这表明利益相关者仍然没有完全理解系统的需求。分析模型及其构建方法将在第8～10章详细说明，下面仅提供简要的概述。

7.5.1 分析模型的元素

有很多不同的方法可用来考察计算机系统的需求。某些软件人员坚持最好选择一个表达模式（例如用例）并排斥所有其他的模式。有些专业人士则相信使用许多不同的表达模式来描述分析模型是值得的，不同的表达模式促使软件团队从不同的角度考虑需求——一种方法更有可能造成需求遗漏、不一致性和歧义性。一些普遍的元素对大多数分析模型来说都是通用的。

基于场景的元素。使用基于场景的方法可以从用户的视角描述系统。例如，基本的用例（7-4节）及其相应的用例图（图7-2）可演化成更精细的基于模板的用例。需求模型的基于场景的元素通常是正在开发的模型的第一部分。同样，它们也作为创建其他建模元素时的输入。例如，图7-3是获取需求并用用例进行表述的UML活动图^①，图中给出了最终基于场景的三层详细表达。

建议 把利益相关者包括进来通常是个好主意。做到这一点最好的方法之一是让每个利益相关者写下描述将如何使用软件的用例。

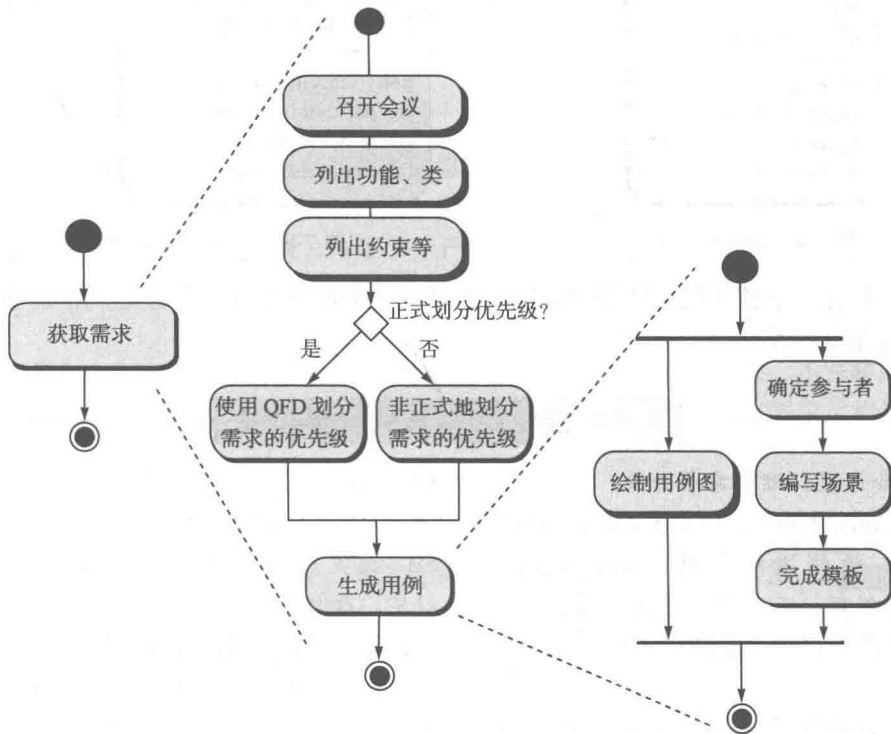


图 7-3 获取需求的 UML 活动图

① 不熟悉 UML 符号的读者请参考附录 1 中的 UML 基本指南。

基于类的元素。每个使用场景都意味着当一个参与者和系统交互时所操作的一组对象，这些对象被分成类——具有相似属性和共同行为的事物集合。例如，可以用UML类图描绘 SafeHome 安全功能的 Sensor 类，如图 7-4 所示。注意，UML 类图列出了传感器的属性（如 name、type）和可以用于修改这些属性的操作（如 identify、enable）。除了类图，其他分析建模元素描绘了类之间的协作以及类之间的关联和交互。在第 9 章中将有更详细的讨论。

行为元素。基于计算机的系统行为能够对所选择的设计和所采用的实现方法产生深远的影响。因此，需求分析模型必须提供描述行为的建模元素。

状态图是一种表现系统行为的方法，该方法描绘系统状态以及导致系统改变状态的事件。状态是任何可以观察到的行为模式。另外，状态图还指明了在某个特殊事件后采取什么动作（例如激活处理）。

为了更好地说明状态图的使用，考虑将软件嵌入 SafeHome 的控制面板，并负责读取用户的输入信息。简化的 UML 状态图如图 7-5 所示。

建议 一种分离类的方法是查找用例脚本中的叙述性名词。至少某些名词将是候选类。第 11 章中将详细说明这一点。

关键点 状态是外部可观察到的行为模式，外部激励导致状态间的转换。

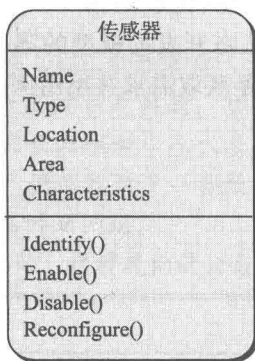


图 7-4 Sensor 类图

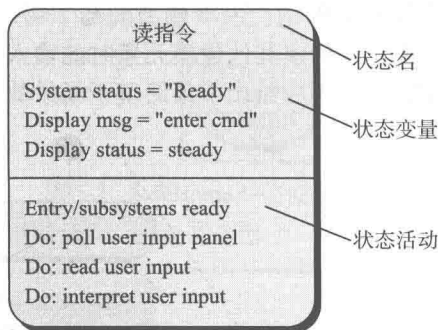


图 7-5 UML 状态图表示

另外，作为一个整体的系统行为表述也能够建模于各个类的行为之上。第 10 章将有更多关于行为建模的讨论。

SafeHome 初步的行为建模

[场景] 会议室，继续需求会议。

[人物] Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人：我们刚才差不多已经讨论完了 SafeHome 的住宅安全功能。但是在结束之前，我希望讨论一下功能的行为。

营销人员：我不太理解你所说的行为意味着什么。

Ed（大笑）：那就是如果产品行为错误就让它“暂停”。

主持人：不太准确，让我解释一下。

（主持人向需求收集团队解释行为建模的基本知识。）

营销人员：这看起来有点技术性，我不敢确定能不能在这里帮上忙。

主持人：你当然可以。你从用户的角度观察到什么行为？

营销人员：嗯……好的，系统将监控传感器、从房主那里读指令，还将显示其状态。

主持人：看到了吧，你是可以帮上忙的。

Jamie：还应该使用计算机确定是否有任何

输入，例如基于 Internet 的访问或配置信息。

Vinod：是的，实际上，配置系统是其权利内的一个状态。

Doug：你这家伙开始转过弯儿了，让我们多想一些……有方法把这个画出来吗？

主持人：有方法，但等到会后再开始吧。

7.5.2 分析模式

任何有一些软件项目需求工程经验的人都开始注意到，在特定的应用领域内某些事情在所有的项目中重复发生^①。这些分析模式 [Fow97] 在特定应用领域内提供一些解决方案（如类、功能、行为），在为许多应用项目建模时都可以重复使用。

Geyer-Schulz 和 Hahsler[Gey01] 提出了使用分析模式的两个优点：

首先，分析模式提高了抽象分析模型的开发速度，通过提供可重复使用的分析模型捕获具体问题的主要需求，例如关于优点和约束的说明。其次，通过建议的设计模式和可靠的通用问题解决方案，分析模式有利于把分析模型转化为设计模型。

通过参照模式名称可把分析模式整合到分析模型中。同时，这些分析模式还将存储在仓库中，以便需求工程师能通过搜索工具发现并应用它们。在标准模板 [Gey01]^② 中会提供关于分析模式（和其他类型模式）的信息。分析模式的样例和有关这一论题更多的讨论在第 10 章中。

建议 如果你想更快获得用户需求并为你的团队提供已经证实可用的方法，那就使用分析模式。

7.5.3 敏捷需求工程

敏捷需求工程的意图是把利益相关者的思想传递给软件团队，而不是生成扩展的分析工作产品。在许多情况下，需求未被预定义，但可作为每次产品迭代开发的开始。当敏捷团队深入地理解了产品的关键特性时，与下一个产品的增量相关的用户故事（第 5 章）便可得到精炼。敏捷过程鼓励尽早定义和实施优先级最高的产品特性，这样能尽早生成并测试工作原型。

敏捷需求工程涉及软件项目中一些常见的重要问题：需求高发散性，不完整的开发技术知识，客户在看到工作原型之前不能清晰表达他们的愿景。敏捷过程将需求过程和设计活动分离开来。

7.5.4 自适应系统的需求

自适应系统^③能自我调整配置、增加功能、自我保护并从失效中恢复，而且，在完成这些活动时，其内部复杂性是对用户隐藏的 [Qur09]。自适应需求阐明了自适应系统的各种必备的变化性。当在同一时间指定软件产

提问 自适应系统的特点是什么？

① 在某些情况下，事情会重复发生而不论应用领域是什么。例如，不管所考虑的应用领域是什么，用户接口的特点和功能都是共有的。

② 文献中已经提出了各种各样的模式模板，感兴趣的读者可以参阅 [Fow97]、[Gam95]、[Yac03] 和 [Bus07]。

③ 自适应系统的一个实例是“位置警告”应用，它根据所在移动平台的位置来自适应地调整系统的行为。

品的一种功能或质量表现时，这意味着需求中必定包含着变化性或灵活性的概念。变化性可能包括时间的不确定性、用户角色的差别（例如最终用户与系统管理员的差别）、基于问题域的行为扩展（例如商业或教育）或利用系统资产预定义行为。

捕获自适应需求时所关注的问题与传统系统中需求工程所关注的问题相同。然而，在逐一回答这些问题时，可将其表示成特定的变化性。答案变化越大，结果系统的复杂性越大，这样才能容纳这些需求。

7.6 避免常见错误

Buschmann[Bus10] 描述了作为软件团队实施需求工程时必须避免的三个相关错误，即对特性、灵活性和性能的过分偏好。

特性偏好是指以功能覆盖率来表征整体系统质量的做法。某些组织倾向于尽可能提早交付等量的功能，从而保证最终产品的整体质量。认为越多越好的商务利益相关者会参与驱动这些事。还有一种软件开发者倾向快速实施简易功能，而不考虑它们的质量。事实上软件项目失败的最常见原因之一是缺少可实用的质量——而不是丢失功能。为了避免落入这个陷阱，你应与其他利益相关者讨论系统必需的关键功能，确保每项已交付的功能都具备了所有必要的质量特性。

灵活性偏好发生在软件工程师过分重视产品的自适应性和配置便利性时。过分灵活的系统会很难配置，并且可操作性差，这是系统范围定义混乱的征兆。然而根本原因可能是开发者使用灵活性来应对不确定性，而不是尽早定稿设计方案。他们提供设计“钩子”，从而允许增加计划外的特性。结果是“灵活”系统产生了不必要的复杂性，越难测试就会有越多的管理挑战。

性能偏好是指软件开发者过分关注质量特性的方面的系统性能开销，如可维护性、可靠性和安全性。系统性能特性应该部分取决于非功能软件需求的评估。性能应该与产品的商业需求一致，同时必须与其他系统特性相兼容。

习题与思考题

- 7.1 为什么大量的软件开发人员没有足够重视需求工程？以前有没有什么情况让你可以跳过需求工程？
- 7.2 你负责从一个客户处获取需求，而他告诉你太忙了没时间见面，这时你该怎么做？
- 7.3 讨论一下当需求必须从三四个不同的客户中提取时会发生什么问题。
- 7.4 为什么我们说需求模型表现了系统的时间快照？
- 7.5 让我们设想你已经说服客户（你是一个绝好的销售人员）同意你作为一个开发人员所提出来的每一个要求，这能够让你成为一个高明的协商人员吗？为什么？
- 7.6 想出 3 个以上在需求起始阶段可能要问利益相关者的“与环境无关的问题”。
- 7.7 开发一个促进需求收集的“工具包”。工具包应包含：一系列需求收集会议的指导原则，用于协助创建列表的材料，以及其他任何可能有助于定义需求的条款。
- 7.8 你的指导老师将把班级分成 4 或 6 人的小组，组中一半的同学扮演市场部的角色，另一半将扮演软件工程部的角色。你的工作是定义本章所介绍的 SafeHome 安全功能的需求，并使用本章所提出的指导原则引导需求收集会议。
- 7.9 为如下活动之一开发一个完整的用例：
 - a. 在 ATM 提款。

- b. 在餐厅使用信用卡付费。
 - c. 使用一个在线经纪人账户购买股票。
 - d. 使用在线书店搜索书（某个指定主题）。
 - e. 你的指导老师指定的一个活动。
- 7.10 用例“异常”代表什么？
- 7.11 选取一个问题 7.9 中列举的活动，写一个用户故事。
- 7.12 考虑问题 7.9 中你生成的用户用例，为应用系统写一个非功能性需求。
- 7.13 用你自己的话描述一个分析模式。
- 7.14 使用 7.5.2 节描述的模板，为下列应用领域建议一个或多个分析模式：
- a. 会计软件
 - b. E-mail 软件
 - c. 互联网浏览器
 - d. 字符处理系统
 - e. 网站生成系统
 - f. 由指导老师特别指定的应用领域
- 7.15 在需求工程活动的协商情境中，“双赢”意味着什么？
- 7.16 你认为当需求确认揭示了一个错误时将发生什么？谁将参与错误修正？
- 7.17 哪 5 个任务组成了综合需求监控程序？

扩展阅读与信息资源

因为需求工程是成功创建任何复杂的基于计算机的系统的核心，所以大量的书籍都在讨论需求工程。Chemuturi（《Requirements Engineering and Management of Software Development Projects》，Springer，2013）阐明了需求工程的重要部分，Pohl 和 Rupp（《Requirements Engineering Fundamentals》，Rocky Nook，2011）表述了基本原理和观念，Pohl（《Requirements Engineering》，Springer，2010）提供了一个完整的需求工程流程的详细视图。Young（《The Requirements Engineering Handbook》，Artech House Publishers，2003）对需求工程任务进行了更深层次的讨论。

Beaty 和 Chen（《Visual Models for Software Products Best Practices》，Microsoft Press，2012），Robertson（《Mastering the requirements Process: Getting Requirements Right》，3rd ed.，Addison-Wesley，2012），Hull 和她的同事（《Requirements Engineering》，3rd ed.，Springer-Verlag，2010），Bray（《An Introduction to Requirements Engineering》，Addison-Wesley，2002），Arlow（《Requirements Engineering》，Addison-Wesley，2001），Gilb（《Requirements Engineering》，Addison-Wesley，2000），Graham（《Requirements Engineering and Rapid Development》，Addison-Wesley，1999），Sommerville 和 Kotonya（《Requirement Engineering: Processes and Techniques》，Wiley，1998），等等，他们都讨论了需求工程这一主题。Wiegers（《More About Software Requirements》，Microsoft Press，2010）提供了很多需求收集和管理的实践。

Withall（《Software Requirement Patterns》，Microsoft Press，2007）描述了基于模式视图的需求工程。Ploesch（《Contracts, Scenarios and Prototypes》，Springer-Verlag，2004）讨论了开发软件需求的先进技术。Windle 和 Abreo（《Software Requirements Using the Unified Process》，Prentice-Hall，2002）从统一过程和 UML 符号的角度讨论了需求工程。Alexander 和 Steven（《Writing Better Requirements》，Addison-Wesley，2002）提出了一套简短的指导原则，目的是编写清楚的需求，使用场景表现需求并

评审最终结果。

用例建模通常在创建分析模型的所有其他方面时使用，讨论该主题的资料有：Rosenberg 和 Stephens (《Use Case Driven Object Modeling with UML: Theory and Practice》，Apress, 2007)，Denny (《Succeeding with Use Cases: Working Smart to Deliver Quality》，Addison-Wesley, 2005)，Alexander 和 Maiden (eds.) (《Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle》，Wiley, 2004)。Leffingwell 和他的同事 (《Managing Software Requirements: A Use Case Approach》，2nd ed., Addison-Wesley, 2003) 表述了非常有用的需求收集的最佳实践。

有些书讨论了敏捷需求，包括：Adzic (《Specification by Example: How Successful Teams Deliver the Right Software》，Manning Publications, 2011)，Leffingwell (《Agile Requirements: Lean Requirements for Teams, Programs, and Enterprises》，Addison-Wesley, 2011)，Cockburn (《Agile Software Development: The Cooperative Game》，2nd ed., Addison-Wesley, 2006)，Cohn (《User Stories Applied: For Agile Software Development》，Addison-Wesley, 2004)。

网上有大量丰富的关于需求工程和分析的信息资源。与需求工程和分析相关的最新参考文献可以在 SEPA 网站 <http://www.mhhe.com/pressman> 上找到。

需求建模：基于场景的方法

要点浏览

概念：文字记录是极好的交流工具，但并不一定是表达计算机软件需求的最好方式。需求建模使用文字和图表的综合形式，以相对容易理解的方式描绘需求，更重要的是，可以更直接地评审它们的正确性、完整性和一致性。

人员：软件工程师（有时被称作分析师）使用从客户那里获取的需求来构建模型。

重要性：为了确认软件需求，你需要从不同的视角检验需求。本章将从基于场景的观点考虑需求建模并检验如何在 UML 中使用补充场景。在第 9 章和第 10 章会学习需求建模的其他“维度”。在检验大量不同维度时，会增加发现错误的概率。这些错误可能是与表象不一致，或可能是没有发现的缺失项。

步骤：基于场景的建模从用户的角度表现系统。在基于场景建模时，将更好地理解用户如何与软件交互，发现没有覆盖到的利益相关者所需的主要系统功能和特性。

工作产品：基于场景建模产生的面向文本的表达称作“用例”。用例描述了特定交互方式，形成非正式（只简单描述）或更加结构化或正规化的自然特征。这些用例能补充大量不同的 UML 图，覆盖更多交互的程序化观点。

质量保证措施：必须评审需求建模工作产品的正确性、完整性和一致性，必须反映所有利益相关者的要求并为从中导出设计建立基础。

在技术层面上，软件工程开始于一系列的建模工作，最终生成待开发软件的需求规格说明和设计表示。需求模型实际上是一组模型^①，是系统的第一个技术表示。

在一本关于需求建模方法的开创性书籍中，Tom DeMarco[Dem79] 如下这样描述该过程：

回顾分析阶段的问题和过失，我建议对分析阶段的目标进行以下的增补。分析的结果必须是高度可维护的，尤其是要将此结果应用于目标文档（软件需求规格说明）。必须使用一种有效的分割方法解决规模问题，维多利亚时代小说式的规格说明是不行的。尽可能使用图形符号。考虑问题时必须区分逻辑的（本质）和物理的（实现）……无论如何，我们至少需要……某种帮助我们划分需求的方法，并在规格说明前用文档记录该划分……某种跟踪和评

关键概念

活动图
域分析
正式用例
需求分析
需求建模
基于场景建模
泳道图
UML 模型
用例
用例异常

① 本书过去的版本使用分析模型这个术语而不是需求模型。本版中决定使用这两个术语，以便表达在解决问题的不同方面时定义的建模活动。分析是获取需求时的动作。

估接口的手段……使用比叙述性文本更好的新工具来描述逻辑和策略……

尽管 DeMarco 在 25 年前就写下了关于分析建模的特点，但他的意见仍然适用于现代的需求建模方法和表示方法。

8.1 需求分析

需求分析产生软件工作特征的规格说明，指明软件和其他系统元素的接口，规定软件必须满足的约束。在需求分析过程中，软件工程师（有时这个角色也被称作分析师或建模师）可以细化在前期需求工程的起始、获取、协商任务中建立的基础需求（第 7 章）。

需求建模动作结果为以下一种或多种模型类型：

- 场景模型：出自各种系统“参与者”观点的需求。
- 面向类的模型：表示面向对象类（属性和操作）的模型，其方式为通过类的协作获得系统需求。
- 基于行为和模式的模型：描述如何将软件行为看作外部“事件”后续的模式。
- 数据模型：描述问题信息域的模型。
- 面向流的模型：表示系统的功能元素并且描述当功能元素在系统中运行时怎样进行数据变换。

这些模型为软件设计者提供信息，这些信息可以被转化为结构、接口和构件级的设计。最终，在软件开发完成后，需求模型（和需求规格说明）就为开发人员和客户提供了评估软件质量的手段。

本章关注基于场景的建模，这项技术在整个软件工程界发展迅猛。在第 9 章和第 10 章，我们考虑基于类的模型和行为模型。在过去十几年，人们已经不常使用流和数据建模，而逐步流行使用场景和基于类的方法，以此作为行为方法和基于模式技术的补充。^①

8.1.1 总体目标和原理

在整个分析建模过程中，软件工程师的主要关注点集中在做什么而不是怎么做。在特定环境下发生哪些用户交互？系统处理什么对象？系统必须执行什么功能？系统展示什么行为？定义什么接口？有什么约束？^②

在前面的章节中，我们注意到在该阶段要得到完整的需求规格说明是不可能的。客户也许无法精确地确定想要什么，开发人员也许无法确定能恰当地实现功能和性能的特定方法，这些现实情况都削弱了迭代需求分析和建模方法的效果。分析师将为已经知道的内容建模，并使用该模型作为软件进一步扩展的设计基础。^③

需求模型必须实现三个主要目标：（1）描述客户需要什么；（2）为软件设计奠定基础；

引述 任何一个需求“视图”都不足以理解和描述一个复杂系统所需的行为。

Alan M. Davis

关键点 一旦软件完成后，分析模型和需求规格说明书将成为评估软件质量的手段。

引述 需求不是架构。需求既不是设计，也不是用户接口。需求就是指需要什么。

Andrew Hunt,
David Thomas

① 在这一版本中我们省略了面向流建模和数据建模。但是在网上可以找到这些较老的需求建模方法的大量信息。如果你感兴趣，可以搜索关键词“结构化分析”进行查找。

② 应该注意，当客户变得更加精通技术时，规格说明书中的“怎么做”需同“做什么”一样重要。但是，基本关注点应保留在“做什么”上。

③ 软件团队也可以花些功夫选择生成一个原型（第 4 章），以便更好地理解系统的需求。

(3) 定义在软件完成后可以被确认的一组需求。分析模型在系统级描述和软件设计(第11~14章)之间建立了桥梁。这里的系统级描述给出了在软件、硬件、数据、人员和其他系统元素共同作用下的整个系统或商业功能,而软件设计给出了软件的应用程序结构、用户接口以及构件级的结构。这个关系如图8-1所示。

重要的是要注意需求模型的所有元素都可以直接跟踪到设计模型。通常难以清楚地区分这两个重要的建模活动之间的设计和分析工作,有些设计总是作为分析的一部分进行,而有些分析将在设计中进行。

8.1.2 分析的经验原则

Arlow 和 Neustadt[Arl02] 提出了大量有价值的经验原则,在创建分析模型时应该遵循这些经验原则:

- 模型应关注在问题域或业务域内可见的需求,抽象的级别应该相对高一些。“不要陷入细节”[Arl02],即不要试图解释系统将如何工作。
- 需求模型的每个元素都应能增加对软件需求的整体理解,并提供对信息域、功能和系统行为的深入理解。
- 关于基础结构和其他非功能的模型应推延到设计阶段再考虑。例如,可能需要一个数据库,但是只有在已经完成问题域分析之后才应考虑实现数据库所必需的类、访问数据库所需的功能以及使用时所表现出的行为。
- 最小化整个系统内的关联。表现类和功能之间的联系非常重要,但是,如果“互联”的层次非常高,则应该想办法减少互联。
- 确认需求模型为所有利益相关者都带来价值。对模型来说,每个客户都有自己的使用目的。例如,业务人员将使用模型确认需求,设计人员将使用模型作为设计的基础,质量保证人员将使用模型帮助规划验收测试。
- 尽可能保持模型简洁。如果没有提供新的信息,就不要添加附加图表;如果一个简单列表够用,就不要使用复杂的表示方法。

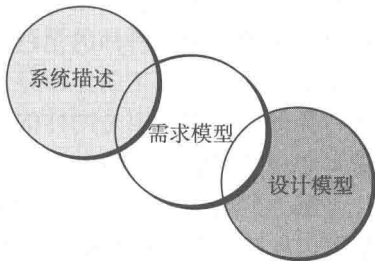


图 8-1 需求模型在系统描述和设计模型之间建立桥梁

关键点 分析模型应该描述什么是客户所需,应该建立设计的基础,建立有效的目标。

提问 进行需求分析时有没有可以帮助我们指南?

引述 值得进攻的问题总是通过反击证明其价值。
Piet Hein

8.1.3 域分析

在需求工程讨论中(第7章),我们注意到分析模式通常在特定业务领域内的很多应用系统中重复发生。如果用一种方式对这些模式加以定义和分类,让软件工程师或分析师识别并复用这些模式,将促进分析模型的创建。更重要的是,应用可复用的设计模式和可执行的软件构件的可能性将显著增加。这将把产品投放市场的时间提前,并减少开发费用。

但问题是,首先如何识别分析模式?由谁来对分析模式进行定义和分类,并为随后的项目准备好分析模式?这些问题的答案在域分析中。Firesmith [Fir93] 这样描述域分析:

软件域分析是指识别、分析和详细说明某个特定应用领域的共同需求,特别是那些在该应用领域内被多个项目重复使用的……(面向对象的域分析是)在某个特定应用领域内,根

网络资源 很多关于域分析的有用信息可以在 www.sei.cmu.edu 中找到。

据通用的对象、类、部件和框架，识别、分析和详细说明公共的、可复用的能力。

“特定应用领域”的范围从航空电子设备到银行业，从多媒体视频游戏到医疗设备中的嵌入式软件。域分析的目标很简单：查找或创建那些广泛应用的分析类或分析模式，使其能够复用。^①

使用本书前面介绍的术语，域分析可以被看作软件过程的一个普适性活动。意思是域分析是正在进行的软件工程活动，而不是与任何一个软件项目相关的。域分析师的角色有些类似于重型机械制造业中一名优秀的刀具工的角色。刀具工的工作是设计并制造工具，这些工具可被很多人用来进行类似的而不一定是同样的工作。域分析师^②的角色是发现和定义可复用的分析模式、分析类和相关信息，这些也可用于类似但不要求必须是完全相同的应用。

图 8-2[Arn89] 说明了域分析过程的关键输入和输出。应该调查领域知识的来源以便确定可以在整个领域内复用的对象。

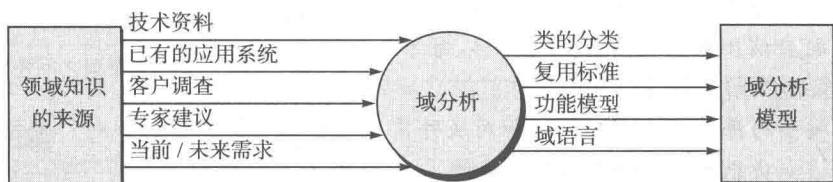


图 8-2 域分析的输入和输出

关键点 域分析不关注特定的应用系统，而是关注应用所属的领域。其目的在于识别那些解决可用于域内所有应用系统的共同问题。

SafeHome | 域分析

[场景] Doug Miller 的办公室，在销售业务会议之后。

[人物] Doug Miller，软件工程经理；Vinod Raman，软件工程团队成员。

[对话]

Doug：我需要你做一个特殊的项目，Vinod。我将会把你调离需求收集会议。

Vinod（皱眉不悦）：太糟了。这可行吗……我已从中获取了一些需求。出什么事啦？

Doug：Jamie 和 Ed 将接管你的工作。不管怎样，市场部坚持要我们在第一次发布的 SafeHome 中交付具有互联网能力的家庭安全功能。我们一直紧张地为此工作着……

没有足够的时间和人力，所以我们马上要解决 PC 接口和 Web 接口两个问题。

Vinod（看上去很疑惑）：我不知道原先设定的计划……我们甚至还没有完成需求收集。

Doug（无精打采地微笑）：我知道，但时间太紧了，我决定马上和市场部开始战略合作……无论如何，一旦从所有需求收集会议上获得信息，我们就将重新审视任何不确定的计划。

Vinod：好的，会发生什么事？你要我做什么事吗？

Doug：你知道“域分析”吗？

Vinod：略知一些。在建立应用系统时为

① 域分析的一个补充观点是：“包括为域建模，因此软件工程师和其他的利益相关者可以更好地学习……不是所有域的类都必然导致可复用的类。” [Let03]。

② 不要认为有域分析员在工作，软件工程师就不需要理解应用问题的领域。软件团队的每个成员都应该一定程度地了解软件将要工作的领域。

做同一件事的应用系统寻找相似的模式。如果可能，可以在工作中剽窃这些模式并复用它们。

Doug：我觉得用剽窃这个词不太恰当，但你的意思基本是正确的。我想让你做的事是开始研究可控制 SafeHome 这类系统的现存用户接口。我想你需要组织一套模式和分析类，它们通常既能坐在房间里处理基于 PC 的接口，也能处理通过互联网进入的基于浏览器的接口。

Vinod：把它们做成相同的东西可以节省时间……为什么不这么做呢？

Doug：哦……. 有你这种想法的人真是非

常好。整个核心要点是如果能识别出两种接口，那么就采用相同的代码，等等，这样我们就节省了时间和人力。这些正是市场部坚持的。

Vinod：那你想要什么？类，分析模式，设计模式？

Doug：所有。非正式地说这就是关键所在。我就是想稍早一些开始我们的内部分析和设计工作。

Vinod：我将在类库中看看我们已经得到了哪些。我也会使用几个月前我读的一本书中的模式模版。

Doug：太好了，继续工作吧。

8.1.4 需求建模的方法

一种考虑数据和处理的需求建模方法称作结构化分析，其中处理过程将数据作为独立实体加以转换。数据对象建模定义了对对象的属性和关系，操作数据对象的处理建模应表明当数据对象在系统内流动时处理过程将如何转换数据。

需求建模的第二种方法称作面向对象的分析，这种方法关注类的定义和影响客户需求的类之间的协作方式。UML 和统一过程（第 4 章）主要是面向对象的分析方法。

在本书这一版中，我们已经选择强调采用面向对象分析的元素进行 UML 建模。目的是建议一种联合的表达方式，给项目利益相关者提供最好的软件需求，以便为软件设计提供桥梁。

如图 8-3 所示，需求模型的每个元素表示源自不同观点的问题。基于场景的元素表述用户如何与系统和使用软件时出现的特定活动序列进行交互。基于类的元素的内容包括：系统操作的对象，应用在这些对象间影响操作和对象间关系（某层级）的操作，以及定义

的类间发生的协作。行为元素描述了外部事件如何改变系统或驻留在系统里的类的状态。最后，面向流的元素表示信息转换的系统，描述了数据对象在流过各种系统功能时是如何转换的。

需求模型导出每个建模元素的派生类。然而，每个元素（即用于构建元素和模型的图表）的特定内容可能因项目而异。就像我们在本书中多次提到的那样，软件团队必须想办法保持模型的简单性。只有那些为模型增加价值的建模元素才能使用。

引述 分析容易使人灰心丧气，全都是非常复杂的人际关系，不确定且困难的东西。总而言之，分析让人着迷。一旦沉迷，原来轻松构建系统的快乐将难以令你满足。

Tom DeMarco

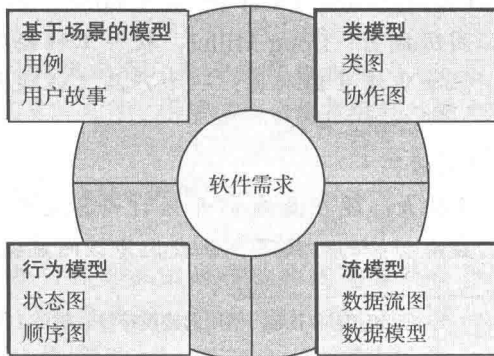


图 8-3 需求模型的元素

8.2 基于场景建模

尽管可以用多种方式度量基于计算机的系统或产品的成果，但用户的满意度仍是其中最重要的。如果软件工程师了解最终用户（和其他参与者）希望如何与系统交互，软件团队将能够更好、更准确地刻画需求特征，完成更有针对性的分析和设计模型。因此，使用 UML^①需求建模将从开发用例、活动图和泳道图形式的场景开始。

8.2.1 创建初始用例

Alistair Cockburn 刻画了一个名为“合同行为”的用例 [Coc01b]。我们在第 7 章讨论的“合同”定义了一个参与者^②使用基于计算机的系统完成某个目标的方法。本质上用例捕获了信息的产生者、使用者和系统本身之间发生的交互。在本节，我们研究如何开发用例，这是分析建模活动的一部分^③。

第 7 章中我们已经提到，用例从某个特定参与者的角度出发，采用简明的语言描述一个特定的使用场景。但是我们如何知道：（1）编写什么？（2）写多少？（3）编写说明应该多详细？（4）如何组织说明？如果想让用例像一个需求建模工具那样提供价值，那么必须回答这些问题。

编写什么？两个首要的需求工程工作——起始和获取——提供了开始编写用例所需要的信息。运用需求收集会议、质量功能部署（Quality Function Deployment, QFD）和其他需求工程机制确定利益相关者，定义问题的范围，说明整体的运行目标，建立优先级顺序，概述所有已知的功能需求，描述系统将处理的信息（对象）。

开始开发用例时，应列出特定参与者执行的功能或活动。这些可以借助所需系统功能的列表，通过与利益相关者交流，或通过评估活动图（作为需求建模中的一部分而开发）获得（8.3.1 节）。

提问 在描述需求模型时常用哪些不同的观点？

引述（用例）只是帮助定义系统之外（参与者）存在什么以及系统应完成什么（用例）。

Ivar Jacobson

关键点 在某些情况下，用例成为最主要的需求工程机制，但是这并不意味着你应该放弃适用的其他建模方法。

SafeHome | 开发另一个初始用户场景

[场景] 会议室，第二次需求收集会议中。

[人物] Jamie Lazar 和 Ed Robbins，软件团队成员；Doug Miller，软件工程经理；三个市场营销人员；一个产品工程代表；一个会议主持人。

[对话]

主持人：现在是我们开始讨论 SafeHome 监视功能的时候了，让我们为访问监视功

能开发一个用户场景。

Jamie：谁在其中扮演参与者的角色？

主持人：我想 Meredith（市场营销人员）已经在该功能上进行了一些工作，你来试试这个角色吧。

Meredith：你想采用我们上次用的方法，是吗？

主持人：是的，同样的方法。

① UML 是本书通篇使用的建模符号。附录 1 为那些不熟悉 UML 基本符号的读者提供了简要指南。

② 参与者不是一个确定的人员，而是人员（或设备）在特定的环境内所扮演的一个角色，参与者“呼叫系统并由系统提供一种服务”[Coc01b]。

③ 用例是用户接口的分析建模中特别重要的一部分，我们将在第 14 章详细讨论接口分析。

Meredith：好的，很明显，开发监视功能的理由是允许房主远距离检查房屋、记录并回放捕获的录像……就是这样。

Ed：我们采用压缩的方法存储图像吗？

主持人：好问题，但是我们现在先不考虑实现的问题，Meredith，你说呢？

Meredith：好的，这样对于监视功能基本上就有两部分……第一部分是配置系统，包括布置建筑平面图——我们需要工具来帮助房主做这件事，第二部分是实际的监视功能本身。因为布局是配置活动的一部分，所以我将重点集中在监视功能。

主持人（微笑）：抢先说出我想说的话。

Meredith：哦……我希望通过电脑或通过 Internet 访问监视功能。我的感觉是 Internet 访问可能使用的频率更高一些。

不管怎样，我希望能够在计算机上和控制面板上显示摄像机图像并移动某个摄像机镜头。在房屋平面设计图上可以选择指定摄像机，我希望可以有选择地记录摄像机输出和回放摄像机输出，我还希望能够使用特殊的密码阻止对某个或多个摄像机的访问。希望有支持小窗口显示形式的选项，即从所有的摄像机显示图像，并能够选择某一个进行放大。

Jamie：那些叫作缩略视图。

Meredith：对，然后我希望从所有摄像机获得缩略视图。我也希望监视功能的接口和所有其他的 SafeHome 接口有相同的外观和感觉。

主持人：干得好，现在，让我们更详细地讨论这个功能……

上面讨论的 SafeHome 住宅监视功能（子系统）确定了如下由参与者房主执行的功能（简化列表）：

- 选择将要查看的摄像机。
- 提供所有摄像机的缩略视图。
- 在计算机的窗口中显示摄像机视图。
- 控制某个特定摄像机的镜头转动和缩放。
- 可选择地记录摄像机的输出。
- 回放摄像机的输出。
- 通过 Internet 访问摄像机监视功能。

随着和利益相关者（扮演房主的人）交谈的增多，需求收集团队将为每个标记的功能开发用例。通常，用例首先用非正式的描述性风格编写。如果需要更正式一些，可以使用类似于第7章中提出的某个结构化的形式重新编写同样的用例，在本节的后面我们将进行重新生成。

为了举例说明，考虑“通过互联网访问摄像机监视设备—显示摄像机视图（Access Camera Surveillance-Display Camera Views, ACS-DCV）”功能，扮演参与者房主的利益相关者可能会编写如下说明。

用例：通过互联网访问摄像机监视设备—显示摄像机视图（ACS-DCV）。

参与者：房主。

如果我正在进行远程访问，那么我可以使用任何计算机上的合适的浏览器软件登录 SafeHome 产品网站。输入我的账号和两级密码，一旦被确认，我可以访问已安装的 SafeHome 系统的所有功能。为取得某个摄像机视图，从显示的主功能按钮中选择“监视”，然后选择“选取摄像机”，这时将会显示房屋的平面设计图，之后再选择感兴趣的摄像机。另一种可选方法是，通过选择“所有摄像机”同时从所有的摄像机查看缩略视图快照。当选

择了某个摄像机时，可以选择“查看”，然后以每秒一帧速度显示的图像就可以在窗口中显示。如果希望切换摄像机，则选择“选取摄像机”，这时原来窗口显示的信息消失，并且再次显示房间的平面设计图，然后就可以选择感兴趣的摄像机，以便显示新的查看窗口。

描述性用例的一种表达形式是通过用户活动的顺序序列表现交互，每个行动由声明性的语句表示。再以 ACS-DCV 功能为例，我们可以写成如下形式。

用例：通过互联网访问摄像机监视设备 - 显示摄像机视图 (ACS-DCV)。

参与者：房主。

1. 房主登录 SafeHome 产品网站。
2. 房主输入账号。
3. 房主输入两个密码（每个至少 8 个字符长度）。
4. 系统显示所有的主要功能按钮。
5. 房主从主要功能按钮中选择“监视”。
6. 房主选择“选取摄像机”。
7. 系统显示房屋的平面设计图。
8. 房主从平面设计图中选择某个摄像机图标。
9. 房主选择“视图”按钮。
10. 系统显示一个由摄像机编号确定的视图窗口。
11. 系统在视图窗口内以每秒一帧的速度显示视频输出。

注意，这个连续步骤的陈述没有考虑其他可能的交互（描述更加自由随意而且确实表达了一些其他选择）。这种类型的用例有时被称作主场景 [Sch98a]。

8.2.2 细化初始用例

为了全面理解用例描述功能，对交互操作给出另外的描述是非常有必要的。

因此，主场景中的每个步骤将通过如下提问得到评估 [Sch98a]：

- 在这一步，参与者能做一些其他动作吗？
- 在这一步，参与者有没有可能遇到一些错误条件？如果有可能，这些错误会是什么？
- 在这一步，参与者有没有可能遇到一些其他行为（如由一些参与者控制之外的事件调用）？如果有，这些行为是什么？

这些问题的答案导致创建一组次场景，次场景属于原始用例的一部分，但是表现了可供选择的行为。例如，考虑前面描述的主场景的第 6 和第 7 步。

6. 房主选择“选取摄像机”。
7. 系统显示房屋的平面设计图。

在这一步，参与者能做一些其他动作吗？答案是肯定的。考虑自由随意的操作方式，参与者可以选择同时查看所有摄像机的缩略视图。因此，一个次场景可能是“查看所有摄像机的缩略视图”。

在这一步，参与者有没有可能遇到一些错误条件？作为基于计算机的系统操作，任何数量的错误条件都可能发生。在该语境内，我们仅仅考虑在第 6 和第 7 步中说明的活动的直接错误条件，问题的答案还是肯定的。带有摄像机图标的房屋平面图可能还没有配置过，这样

引述 用例可以应用在许多（软件）过程中，我们感兴趣的是迭代和风险驱动过程。

Geri Schneider,
Jason Winters

提问 在开发场景用例时，如何检查动作的可选过程？

选择“选取摄像机”就导致错误的条件：“没有为该房屋配置平面设计图”^①。该错误条件就成为一个次场景。

在这一步，参与者有没有可能遇到一些其他行为？问题的答案再一次是肯定的。当第6和第7步发生时，系统可能遇到报警。这将导致系统显示一个特殊的报警通知（类型、地点、系统动作），并向参与者提供和报警性质相关的一组操作。因为这个次场景可以在所有的实际交互中发生，所以不会成为 ACS-DCV 用例的一部分。而且，我们将开发一个单独的用例——遇到报警条件——这个用例可以被其他用例引用。

前面段落描述的每种情景都是以客户用例的异常处理为特征的。异常处理描述了这样一种情景（可能是失败条件或参与者选择了替代方案），该场景导致系统展示出某些不同的行为。

提问 什么是用例异常？我们如何决定这些异常的样子？

Cockburn[Coc01b] 推荐使用“头脑风暴”来推动团队合理地完成每个用例中一系列的异常处理。除了本节前面提到了三个常规问题外，还应该研究下面的问题：

- 在这个用例中是否有某些具有“确认功能”的用例出现？包括引用确认功能，以及可能出现的出错条件。
- 在这些用例中是否有支持功能（或参与者）的应答失败？例如，某个用户动作是等待应答，但该功能已经应答超时了。
- 性能差的系统是否会导致无法预期或不正确的用户活动？例如，一个基于 Web 的接口应答太慢，导致用户在处理按钮上已经做了多重选择。这些选择队列最终不恰当地生成了一个出错条件。

其他问题和回答将继续扩充这份列表，使用下面的标准可使这些问题合理化 [Co01b]：用例应该注明异常处理，即如果软件能检测出异常所发生的条件就应该马上处理这个条件。在某些情况下，异常处理可能拖累其他用例处理条件的开发。

8.2.3 编写正式用例

8.2.1 节表述的非正式用例对于需求建模常常是够用的。但是，当用例需要包括关键活动或描述一套具有大量异常处理的复杂步骤时，我们会希望采用更为正式的方法。

在 SafeHome 中的 ACS-DCV 用例把握了正式用例的典型描述要点。在以下的 SafeHome 中：情境目标确定了用例的全部范围。前提条件描述在用例初始化前应该知道哪些信息。触发器确定“用例开始”的事件或条件 [Coc01b]。场景列出参与者和恰当的系统应答所需要的特定活动。异常处理用于细化初始用例时没有涉及的情景（8.2.2 节）。此外还可能包含其他主题，并给出合理的自我解释。

SafeHome 监视的用例模板

用例：通过互联网访问摄像机监视设备—显示摄像机视图（ACS-DCV）。

迭代：2。最新更改记录：V. Raman，1 月 14 日。

主参与者：房主。

情境目标：从任何远程地点通过互联网查看遍布房间的摄像头输出。

前提条件：必须完整配置系统；必须获得

^① 在该例子中，另一个参与者——系统管理员必须配置平面设计图、安装并初始化（如分配设备编号）所有的摄像头，并且通过系统平面设计图访问和测试每个摄像头能达到的特定作用。

正确的账号和密码。

触发器：房主在远离家的时候决定查看房屋内部。

场景：

1. 房主登录 SafeHome 产品网站。
2. 房主输入他的账号。
3. 房主输入两个密码（每个都至少有 8 个字符的长度）。
4. 系统显示所有的主要功能按钮。
5. 房主从主要功能按钮中选择“监视”。
6. 房主选择“选取摄像机”。
7. 系统显示房屋的平面设计图。
8. 房主从房屋的平面设计图中选择某个摄像机的图标。
9. 房主选择“视图”按钮。
10. 系统显示一个由摄像机编号确定的视图窗口。
11. 系统在视图窗口中以每秒一帧的速度显示视频输出。

异常处理：

1. 账号或密码不正确或不被确认——参看用例“确认账号和密码”。
2. 没有为该系统配置监视功能——系统显示恰当的错误消息，参看用例“配置监视功能”。
3. 房主选择“查看所有摄像机的缩略视图

快照”——参看用例“查看所有摄像机的缩略视图快照”。

4. 平面设计图不可用或是还没有配置——显示恰当的错误消息，参看用例“配置平面设计图”。
5. 遇到报警条件——参看用例“遇到报警条件”。

优先级：必须在基础功能之后实现中等优先级。

何时有效：第三个增量。

使用频率：频率较低。

参与者的连接渠道：通过基于个人计算机的浏览器和互联网连接到 SafeHome 网站。

次要参与者：系统管理员，摄像机。

次要参与者的连接渠道：

1. 系统管理员：基于个人计算机的系统。
2. 摄像机：无线连接。

未解决的问题：

1. 用什么机制保护 SafeHome 产品的雇员在未授权的情况下能使用该功能？
2. 足够安全吗？黑客入侵该功能将使最主要的个人隐私受侵。
3. 在给定摄像机视图所要求的带宽下，可以接受通过互联网的系统响应吗？
4. 若可以使用高带宽的连接，能开发出比每秒一帧更快的视频速度吗？

在很多情况下，不需要创建图形化表示的用户场景。然而，在场景比较复杂时，图表化的表示更有助于理解。正如我们在本书前面所提到的，UML 的确提供了图形化表现用例的能力。图 8-4 为 SafeHome 产品描述了一个初步的用例图，每个用例由一个椭圆表示。本节仅详细讨论了 ACS-DCV。

每种建模注释方法都有其局限性，用例方法也无例外。和其他描述形式一样，用例的好坏取决于它的描述者。如果描述不清晰，用例可能会误导或有歧义。用例关注功能和行为需求，一般不适用于非功能需求。对于必须特别详细和精准的需求建模情境（例如安全关键系统），用例方法就不够用了。

然而，软件工程师遇到的绝大多数情境都适用基于场景建模。如果开发得当，用例作为一个建模工具将带来很多益处。

网络资源 什么时候结束用例编写？关于该话题有价值的论述可以参阅 ootips.org/use-cases-done.html。

8.3 补充用例的 UML 模型

很多基于文本的需求建模情景（即使和用例一样简单）不能简明扼要地传递信息。在这种情况下，你应从大量的 UML 图形模型中进行选择。

8.3.1 开发活动图

UML 活动图在特定场景内通过提供迭代流的图形化表示来补充用例。类似于流程图，活动图使用两端为半圆形的矩形表示一个特定的系统功能，箭头表示通过系统的流，菱形表示分支（标记从菱形发出的每个箭头），实水平线意味着并行发生的活动。ACS-DCV 用例的活动图如图 8-5 所示。应注意到活动图增加了额外的细节，而这些细节是用例不能直接描述的（隐含的）。例如，用户可以尝试有限次数地输入账号和密码，这可以通过“提示重新输入”的判定菱形来体现。

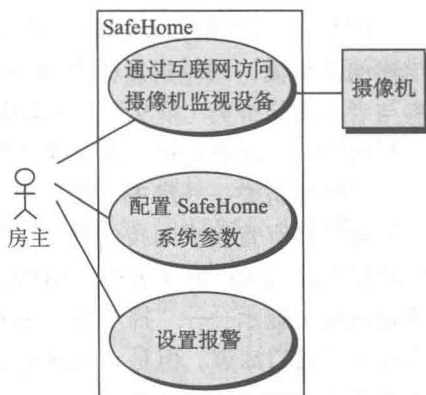


图 8-4 SafeHome 系统的初步用例图

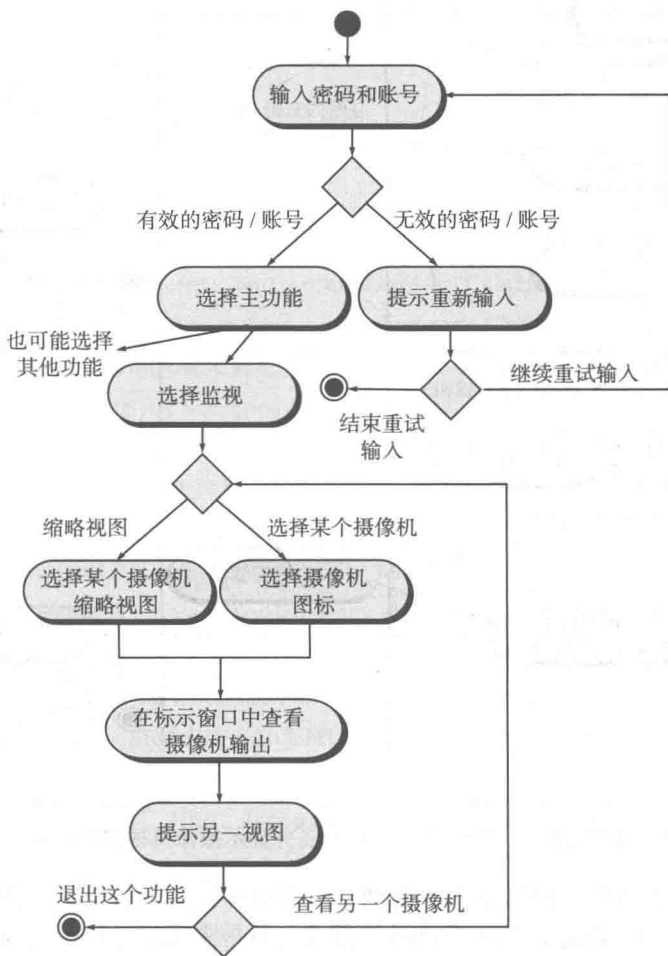


图 8-5 通过互联网访问摄像机监视设备并显示摄像机视图功能的活动图

8.3.2 泳道图

UML 泳道图是活动图的一种有用的变形，允许建模人员表示用例所描述的活动流，同时指出哪个参与者（如果在某个特定用例中涉及了多个参与者）或分析类（第 9 章）负责由活动矩形所描述的活动。职责由纵向分割图中的并行条表示，就像游泳池中的泳道。

三种分析类——房主、摄像机和接口——对于图 8-5 所表示的活动图中的情景具有直接或间接的责任。参看图 8-6，重新排列活动图，和某个特殊分析类相关的活动按类落入相应的泳道中。例如，接口类表示房主可见的用户接口。活动图标记出对接口负责的两个提示——“提示重新输入”和“提示另一视图”。这些提示以及与此相关的判定都落入了接口泳道。但是，从该泳道发出的箭头返回到房主泳道，这是因为房主的活动在房主泳道中发生。

关键点 UML 泳道图表现了活动流和一些判定，并指明由哪个参与者实施。

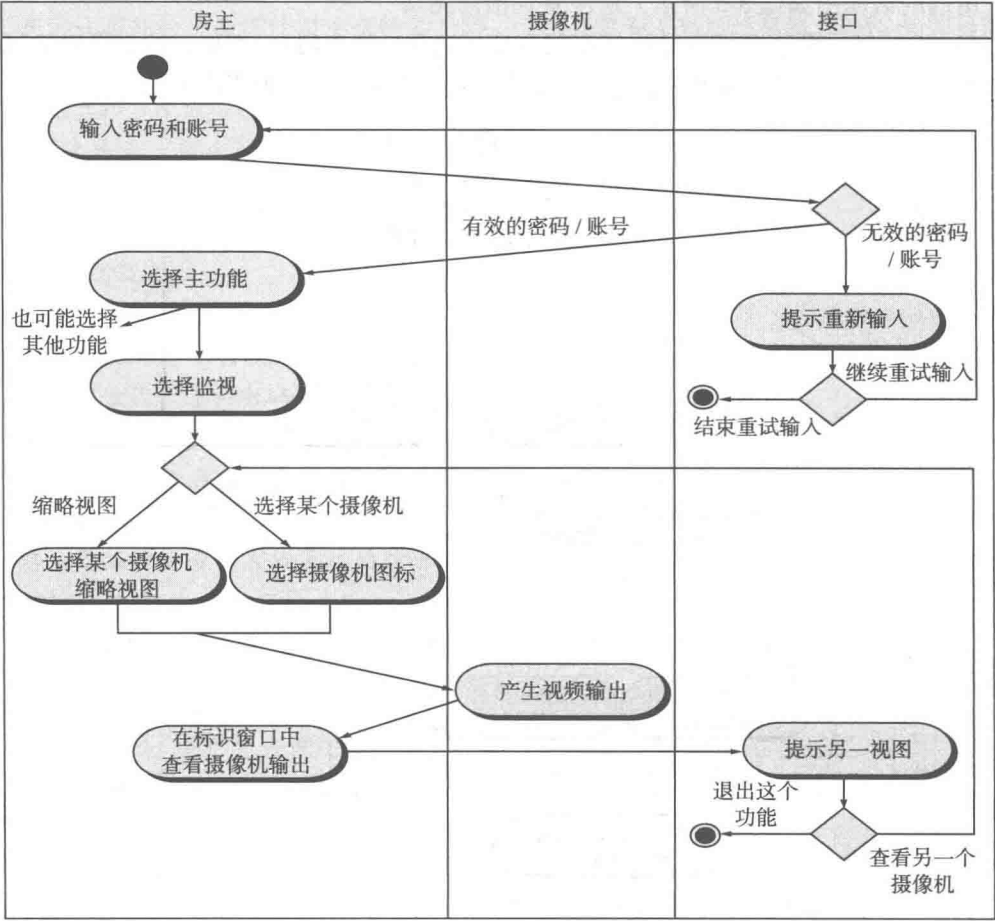


图 8-6 通过互联网访问摄像机监视设备并显示摄像机视图功能的泳道图

借助活动图和泳道图，面向过程的用例表示出各种参与者行使的一些特定功能（或其他处理步骤），以便满足系统需求。但是需求的过程视图仅表示系统的单一维度，在第 9 章和第 10 章，我们将考察需求建模的其他维度。

引述 好模型引导你思考，而坏模型会扭曲它。
Brian Marick

习题与思考题

- 8.1 有没有可能在分析模型创建后立即开始编码？解释你的答案，然后说服反方。
- 8.2 一个单凭经验的分析原则指出模型“应该关注在问题域或业务域中可见的需求”。在这些域中哪些类型的需求是不可见的？提供一些例子。
- 8.3 域分析的目的是什么？如何将域分析与需求模式概念相联系？
- 8.4 有没有可能不完成图 8-3 所示的四种元素就开发出一个有效的分析模型？解释一下。
- 8.5 某个大城市的公共工程部决定开发基于 Web 的路面坑洼跟踪和修补系统（PHTRS）。说明如下：

市民可以登录 Web 站点报告路面坑洼的地点和严重程度。上报后，该信息将记入“路面坑洼跟踪和修补系统”，分配一个标识号，保存如下信息：街道地址、大小（比例从 1 到 10）、位置（中央、路边等）、地区（由街道地址确定）以及修补优先级（由坑洼大小确定）。工作订单数据和每个坑洼有关联，数据包含坑洼位置和大小、维修组标识号、维修组内人员数量、分配的设备、修复耗时、坑洼状态（正在处理中、已修复、临时修复、未修复）、使用的填充材料数量以及修复成本（根据修复耗时、人员数量、材料和使用的设备计算）。最后，生成损失文件以便保存该坑洼所造成的损失报告信息，并包含公民的姓名、地址、电话号码、损失类型、损失金额。PHTRS 基于在线系统，可交互地进行所有查询。

为 PHTRS 系统画出 UML 用例图，你必须对用户和系统的交互方式做一些假设。

- 8.6 编写 2 ~ 3 个用例描述 PHTRS 系统中的各种参与者的角色。
- 8.7 为 PHTRS 系统的某个部分开发一个活动图。
- 8.8 为 PHTRS 系统的一个或多个部分开发一个泳道图。

扩展阅读与信息资源

用例是为所有需求建模方法服务的基础。讨论这一主题的书很多，包括：Gomaa（《Software Modeling: UML, Use Case, Patterns, and Architecture》，Cambridge University Press, 2011），Rosenberg 和 Stephens（《Use Case Driven Object Modeling with UML: Theory and Practice》，Apress, 2007），Denny（《Succeeding with Use Cases: Working Smart to Deliver Quality》，Addison-Wesley, 2005），Alexander 和 Maiden (eds.)（《Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle》，Wiley, 2004），Bittner 和 Spence（《Use Case Modeling》，Addison-Wesley, 2002），Cockburn[Coc01b]。其他参考资料请关注第 8 章。

UML 建模技术可以应用于分析和设计，讨论这方面的书包括：Dennis 和他的同事（《Systems Analysis and Design with UML Version 2.0》，4th ed., Wiley, 2012），O'Docherty（《Object-Oriented Analysis and Design: Understanding System Development with UML2.0》，Wiley, 2005），Arlo 和 Neustadt（《UML 2 and the Unified Process》，2nd ed., Addison-Wesley, 2005），Roques（《UML in Practice》，Wiley, 2004），Larman（《Applying UML and Patterns》，2nd ed., Prentice-Hall, 2001），Rosenberg 和 Scott（《Use Case Driven Object Modeling with UML》，Addison-Wesley, 1999）。

关于需求的书包括 Robertson（《Mastering the Requirements Process: Getting Requirements Right》，3rd ed., Addison-Wesley, 2012），Hull、Jackson 和 Dick（《Requirements Engineering》，3rd ed., Springer, 2010），Alexander 和 Beus Dukic（《Discovering Requirements: How to Specify Products and Services》，Wiley, 2009）。

网上有很多关于需求建模的信息。可以参考 SEPA 网站 www.mhhe.com/pressman 上有关分析建模的最新参考文献。

需求建模：基于类的方法

要点浏览

概念：软件问题总是以一套交互对象为特征，借此在系统内表达每一个感兴趣的事物。每个对象变成对象类中的一个成员。对象的状态描述了每个对象，即数据属性描述了对象。我们可以用基于类的需求建模方法表达上述所有内容。

人员：软件工程师（有时被称作分析师）使用从客户那里获取的需求来建立基于类的模型。

重要性：基于类的需求模型利用客户视角下应用或系统中的对象。这个模型描述了常规客户的系统视图。因此，客户能恰当地进行评估，并尽早获得有用的反馈信息。然后，在重新定义模型时，它将成为软件设计的基础。

步骤：基于类的建模定义了对对象、属性和关系。对一个问题描述做一番简单的探究后，能从问题的陈述中开发外部对象和类，并以基于文本或图表的形式进行表达。在创建了模型的雏形以后，就要对其不断改进，分析和评估其清晰性、完整性和一致性。

工作产品：可以为需求建模选择大量的基于文本和图表形式的分析模型，每种表达方法都提供了一个或多个模型元素的视图。

质量保证措施：必须评审需求建模工作产品的正确性、完整性和一致性。必须反映所有利益相关者的要求并建立一个可以从中导出设计的基础。

关键概念

分析类
分析包
关联
属性
协作性
CRC 建模
依赖性
语法解析
运维
职责

20 世纪 90 年代早期，第一次引入基于类的需求建模方法时，常以面向对象分析作为分类方法。虽然有大量不同的基于类的方法和表达方式，但 Coad 和 Yourdon[Coa91] 为所有人注明了其统一特征：

面向对象方法基于的都是我们最初在幼儿园中学到的内容：对象和属性，全局和部分，类和成员。

在需求建模中使用基于类的方法可以精巧地表达这些常规内容，使非技术型的利益相关者理解项目。随着需求模型的细化和扩展，它还将包含一份规格说明，以帮助软件工程师创建软件的设计部分。

基于类建模表示了系统操作的对象、应用于对象间能有效控制的操作（也称为方法或服务）、这些对象间（某种层级）的关系以及已定义类之间的协作。基于类的分析模型的元素包括类和对象、属性、操作、CRC 模型、协作图和包。下面几节将提供一系列有助于识别和表示这些元素的非正式指导原则。

9.1 识别分析类

当你环顾房间时，就可以发现一组容易识别、分类和定义（就属性和操作而言）的物理

对象。但当你“环顾”软件应用的问题空间时，了解类（和对象）就没有那么容易了。

通过检查需求模型（第8章）开发的使用场景，并对系统开发的用例进行“语法解析”[Abb83]，我们就可以开始进行类的识别了。带有下列线的每个名词或名词词组可以确定为类，将这些名词输入到一个简单的表中，并标注出同义词。如果要求某个类（名词）实现一个解决方案，那么这个类就是解决方案空间的一部分；否则，如果只要求某个类描述一个解决方案，那么这个类就是问题空间的一部分。

一旦分离出所有的名词，我们该寻找什么？分析类表现为如下方式之一。

- 外部实体（例如其他系统、设备、人员）：产生或使用基于计算机系统的信息。
- 事物（例如报告、显示、字母、信号）：问题信息域的一部分。
- 偶发事件或事件（例如所有权转移或完成机器人的一组移动动作）：在系统操作环境内发生。
- 角色（例如经理、工程师、销售人员）：由和系统交互的人员扮演。
- 组织单元（例如部门、组、团队）：和某个应用系统相关。
- 场地（例如制造车间或码头）：建立问题的环境和系统的整体功能。
- 结构（例如传感器、四轮交通工具、计算机）：定义了对应的类或与对象相关的类。

这种分类只是文献中已提出的大量分类之一^①。例如，Budd[Bud96]建议了一种类的分类法，包括数据产生者（源点）、数据使用者（汇点）、数据管理者、查看或观察者类以及帮助类。

还需要特别注意的是：什么不能是类或对象。通常，决不应该使用“命令过程式的名称”为类命名[Cas89]。例如，如果医疗图像系统的软件开发人员使用名字“`InvertImage`”甚至“`ImageInversion`”定义对象，就可能犯下一个小小的错误。从软件获得的 `Image` 当然可能是一个类（这是信息域中的一部分），图像的翻转是适用于该对象的一个操作，很可能将翻转定义为对于对象 `Image` 的一个操作，但是不可能定义单独的类来暗示“图像翻转”。如 Cashman[Cas89] 所言：“面向对象的目的是封装，但仍保持独立的数据以及对数据的操作。”

为了说明在建模的早期阶段如何定义分析类，考虑对 `SafeHome` 安全功能的“处理说明”^②进行语法解析（对第一次出现的名词加下划线，第一次出现的动词采用斜体）。

SafeHome 安全功能 允许房主在安装时配置安全系统，监控所有连接到安全系统的传感器，通过互联网、计算机或控制面板和房主交互信息。

在安装过程中，用 `SafeHome` 个人计算机来设计和配置系统。为每个传感器分配编号和类型，用主密码控制启动和关闭系统，而且当传感器事件发生时会拨打输入的电话号码。

识别出一个传感器事件时，软件激活装在系统上的发声警报，由房主在系统配置活动中指定的延迟时间结束后，软件拨打监控服务的电话号码并提供位置信息，报告检测到的事件性质。电话号码将每隔 20 秒重拨一次，直至电话接通。

引述 真正困难的问题是首先发现什么才是正确的对象（类）。

Carl Argila

提问 分析类如何把自己表现为解决方案空间的元素？

建议 虽然语法解析不能保证万无一失，但如果你正在定义数据对象及其操作的转变，语法解析会让你飞跃上一个非常出色的起始点。

① 另一个重要的分类是指定义实体、边界和控制类，在 9.5 节中讨论。

② “处理说明”类似于用例的风格，但目标稍有不同。“处理说明”提供了将要开发的功能的整体说明，而不是从某个参与者的角度写的场景。但是要注意很重要的一点，在需求收集（获取）部分也会使用语法解析来开发每个用例。

房主通过控制面板、个人计算机或浏览器（统称为接口）来接收安全信息。接口在控制面板、计算机或浏览器窗口中显示提示信息和系统状态信息。房主采用如下形式进行交互活动……

抽取这些名词，可以获得如下表所示的一些潜在类：

潜在类	一般分类	潜在类	一般分类
房主	角色或外部实体	主密码	事物
传感器	外部实体	电话号码	事物
控制面板	外部实体	传感器事件	事件
安装	事件	发声警报	外部实体
系统（别名安全系统）	事物	监控服务	组织单元或外部实体
编号，类型	不是对象，是传感器的属性		

这个表应不断完善，直到已经考虑到了处理说明中所有的名词。注意，我们称列表中的每一输入项为“潜在”对象，在进行最终决定之前还必须对每一项都深思熟虑。

Coad 和 Yourdon[Coa91] 建议了 6 个选择特征，在分析模型中，分析师考虑每个潜在类是否应该使用如下这些特征。

提问 如何确定某个潜在类是否应该真的成为一个分析类？

1. 保留信息。只有记录潜在类的信息才能保证系统正常工作，这样潜在类才能在分析过程中发挥作用。
2. 所需服务。潜在类必须具有一组可确认的操作，这组操作能用某种方式改变类的属性值。
3. 多个属性。在需求分析过程中，焦点应在于“主”信息；事实上，只有一个属性的类可能在设计中有用，但是在分析活动阶段，最好把它作为另一个类的某个属性。
4. 公共属性。可以为潜在类定义一组属性，这些属性适用于类的所有实例。
5. 公共操作。可以为潜在类定义一组操作，这些操作适用于类的所有实例。
6. 必要需求。在问题空间中出现的外部实体，以及任何系统解决方案运行时所必需的生产或消费信息，几乎都被定义为需求模型中的类。

考虑包含在需求模型中的合法类，潜在类应全部（或几乎全部）满足这些特征。判定潜在类是否应包含在分析模型中多少有点主观，而且后面的评估可能会舍弃或恢复某个类。然而，基于类建模的首要步骤就是定义类，因此必须进行决策（即使是主观的）。以此为指导，根据上述选择特征进行了筛选，分析师列出 SafeHome 潜在类，如下表所示。

引述 阶级斗争中，一些阶级胜利了，一些阶级被消灭了……
毛泽东

潜在类	适用的特征编号
房主	拒绝：6 适用，但是 1、2 不符合
传感器	接受：所有都适用
控制面板	接受：所有都适用
安装	拒绝
系统（别名安全系统）	接受：所有都适用
编号，类型	拒绝：3 不符合，这是传感器的属性
主密码	拒绝：3 不符合
电话号码	拒绝：3 不符合
传感器事件	接受：所有都适用
发声警报	接受：2、3、4、5、6 适用
监控服务	拒绝：6 适用，但是 1、2 不符合

应注意到：（1）上表并不全面，必须添加其他类以使模型更完整；（2）某些被拒绝的潜在类将成为被接受类的属性（例如，编号和类型是 Sensor 的属性，主密码和电话号码可能成为 System 的属性）；（3）对问题的不同陈述可能导致做出“接受或拒绝”的不同决定（例如，如果每个房主都有个人密码或通过声音确认，Homeowner 类就有可能接受并满足特征 1 和 2）。

9.2 描述属性

属性描述了已经选择包含在需求模型中的类。实质上，属性定义了类，以澄清类在问题空间的环境下意味着什么。例如，如果我们建立一个系统来跟踪职业棒球手的统计信息，那么类 Player 的属性与用于职业棒球手的养老系统中的属性就是截然不同的。前者，属性可能涉及名字、位置、平均击球次数、担任防守百分比、从业年限、比赛次数等相关信息。后者，以上某些属性仍是有意义的，但另外一些属性将会更换（或扩充），例如，平均工资、充分享受优惠权后的信用、所选的养老计划、邮件地址等。

关键点 属性是在问题的环境下完整定义类的数据对象集合。

为了给分析类开发一个有意义的属性集合，软件工程师应该研究用例并选择那些合理的“属于”类的“事物”。此外，每个类都应回答如下问题：什么数据项（组合项或基本项）能够在当前问题环境内完整地定义这个类？

为了说明这个问题，考虑为 SafeHome 定义 System 类。房主可以配置安全功能以反映传感器信息、报警响应信息、激活或者关闭信息、识别信息等。我们可以用如下方式表现这些组合数据项：

识别信息 = 系统编号 + 确认电话号码 + 系统状态
报警应答信息 = 延迟时间 + 电话号码
激活或者关闭信息 = 主密码 + 允许重试次数 + 临时密码

等式右边的每一个数据项可以进一步地细化到基础级，但是考虑到我们的目标，可以为 System 类组成一个合理的属性列表（图 9-1 中的阴影部分）。

传感器是整个 SafeHome 系统的一部分，但是并没有列出如图 9-1 所示的数据项或属性。已经定义 Sensor 为类，多个 Sensor 对象将和 System 类关联。通常，如果有超过一个项和某个类相关联，就应避免把这个项定义为属性。

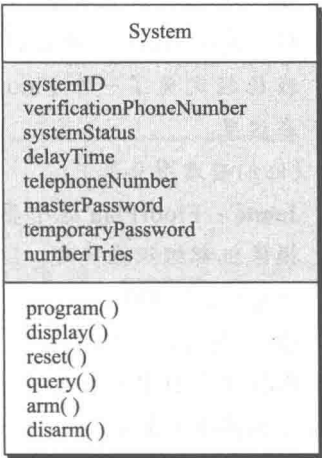


图 9-1 System 类的类图

9.3 定义操作

操作定义了某个对象的行为。尽管存在很多不同类型的操作，但通常可以粗略地划分为 4 种类型：（1）以某种方式操作数据（例如添加、删除、重新格式化、选择）；（2）执行计算的操作；（3）请求某个对象的状态的操作；（4）监视某个对象发生某个控制事件的操作。这些功能通过在属性或相关属性（9.5 节）上的操作来实现。因此，操作必须“理解”类的属性和相关属性的性质。

在第一次迭代要导出一组分析类的操作时，可以再次研究处理说明（或用例）并合理地选择属于该类的操作。为了实现这个目标，可以再次研究语法解析并分离动词。这些动词中的一部分将是合法的操作并能够很容

建议 当为分析类定义操作时，集中于面向问题的行为而不是实现所要求的行为。

易地连接到某个特定类。例如，从本章前面提出的 SafeHome 处理说明中可以看到，“为传感器分配编号和类型”“主密码用于激活和解除系统”，这些短语表明：

- assign() 操作和 Sensor 类相关联。
- program() 操作应用于 System 类。
- arm() 和 disarm() 应用于 System 类。

再进一步研究，program() 操作很可能被划分为一些配置系统所需要的更具体的子操作。例如，program() 隐含着电话号码、配置系统特性（如创建传感器表、输入报警特征值）和输入密码。但是我们暂时把 program() 指定为一个单独的操作。

另外，对于语法解析，分析师能通过考虑对象间所发生的通信获得对其他操作的更为深入的了解。对象通过传递信息与另一个对象通信。在继续对操作进行说明之前，我们探测到了更详实的信息。

SafeHome 类模型

[场景] Ed 的小房间，开始进行分析建模。

[人物] Jamie、Vinod 和 Ed，SafeHome 软件工程团队的成员。

[对话]

Ed 已经从 ACS-DCV（本章前面的 SafeHome 中已有介绍）的用例模板中做了提取类方面的工作，并向他的同事展示了已经提取的类。

Ed：那么当房主希望选择一个摄像机的时候，他必须从一个平面设计图中进行选择。我已经定义了一个 FloorPlan 类，这个图在这里。

（他们查看图 9-2。）

Jamie：FloorPlan 这个类把墙、门、窗和摄像机都组织在一起。这就是那些标记线的意义，是吗？

Ed：是的，它们被称作“关联”，一个类根据我在图中所表示的关联关系和另一个类相关联（在 9.5 节中讨论关联）。

Vinod：那么实际的平面设计图是由墙构成的，并包含摄像机和放置在那些墙中的传感器。平面设计图如何知道在哪里放置那些对象？

Ed：平面设计图不知道，但是其他类知道。例如查看属性 WallSegment，该属性用于

构建墙，墙段（WallSegment）具有起点坐标和终点坐标，其他由 draw() 操作完成。

Jamie：这些也适用于门窗。看起来摄像机有一些额外的属性。

Ed：是的，我要求它们提供转动信息和缩放信息。

Vinod：我有个问题，为什么摄像机有 ID 编号而其他对象没有呢？我注意到有个属性叫 nextWall。WallSegment 如何知道什么是下一堵墙？

Ed：好问题，但正如他们所说，那是由设计决定的，所以我将推迟这个问题直到……

Jamie：让我休息一下……我打赌你已经想出办法了。

Ed（羞怯地笑了笑）：确实，当我们开始设计时我要采用列表结构来建模。如果你坚信分析和设计是分离的，那么我安排的详细程度等级就有问题了。

Jamie：对我而言看上去非常好。只是我还有一些问题。

（Jamie 问了一些问题，因此做了一些小的修改。）

Vinod：你有每个对象的 CRC 卡吗？如果有，我们应该进行角色演练，以确保没有

任何遗漏。

Vinod：这不难，而且确实有用，我给你演示一下。

Ed：我不太清楚如何做。

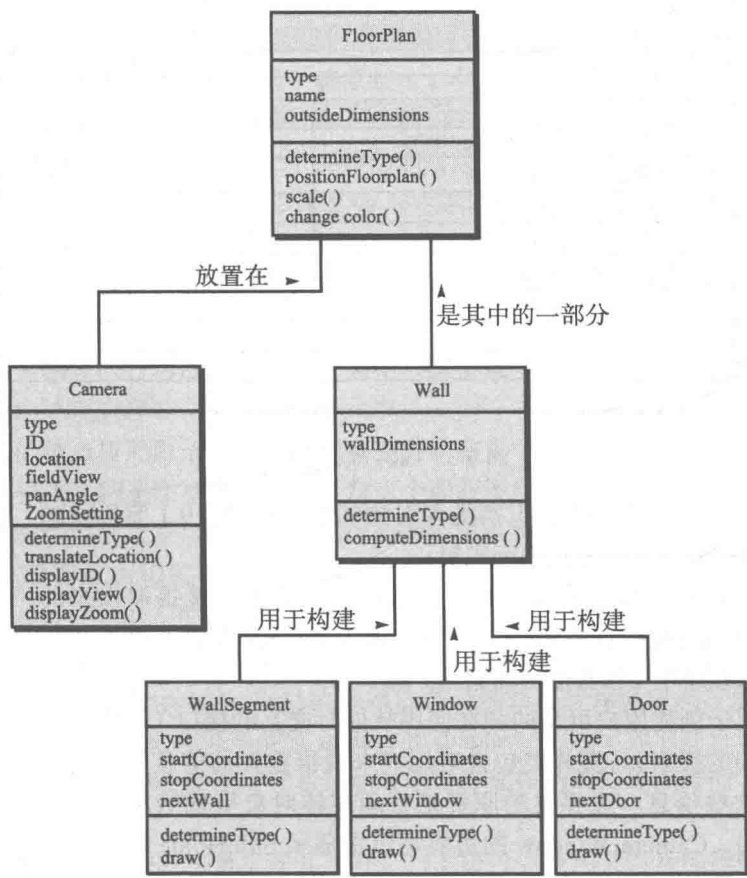


图 9-2 FloorPlan 类的类图

9.4 类 – 职责 – 协作者建模

类 – 职责 – 协作者 (Class-Responsibility-Collaborator, CRC) 建模 [Wir90] 提供了一个简单方法，可以识别和组织与系统或产品需求相关的类。Ambler[Amb95] 用如下文字解释了 CRC 建模：

CRC 模型实际上是表示类的标准索引卡片的集合。这些卡片分三部分，顶部写类名，卡片主体左侧部分列出类的职责，右侧部分列出类的协作者。

事实上，CRC 模型可以使用真实的或虚拟的索引卡，意图是有组织地表示类。职责是和类相关的属性和操作。简单地说，职责就是“类所知道或能做的任何事” [Amb95]。协作者提供完成某个职责所需要信息的类。通常，协作意味着信息请求或某个动作请求。

FloorPlan 类的一个简单 CRC 索引卡如图 9-3 所示。CRC 卡上所列出的职责只是初步的，

引述 使用 CRC 卡的一个目的是尽早舍弃、频繁舍弃以及低成本舍弃。事实上抽出一叠卡片比改编大量源代码要容易得多。

C. Horstmann

可以添加或修改。在职责栏右边的 Wall 和 Camera 是需要协作的类。

类: FloorPlan	
说明	
职责:	协作者:
定义住宅平面图的名称 / 类型	
管理住宅平面图的布局	
缩放显示住宅平面图	
合并墙、门和窗	Wall
显示摄像机的位置	Camera

图 9-3 CRC 模型索引卡

类。在本章前面已经说明了识别类和对象的基本原则。9.1 节所说的类的分类可以通过如下分类方式进行扩展：

- 实体类，也称作模型或业务类，是从问题说明中直接提取出来的（例如 FloorPlan 和 Sensor）。这些类一般代表保存在数据库中和贯穿在应用程序中（除非被明确删除）的事物。
- 边界类用于创建用户可见的和在使用软件时交互的接口（如交互屏幕或打印的报表）。实体类包含对用户来说很重要的信息，但是并不显示这些信息。边界类的职责是管理实体对象呈现给用户的方式。例如，Camera Window 的边界类负责显示 SafeHome 系统监视摄像机的输出。
- 控制类自始至终管理“工作单元”。也就是说，控制类可以管理：（1）实体类的创建或更新；（2）边界类从实体对象获取信息后的实例化；（3）对象集合间的复杂通信；（4）对象间或用户和应用系统间交换数据的确认。通常，直到设计开始时才开始考虑控制类。

职责。在 9.2 节和 9.3 节中已经说明了识别职责（属性和操作）的基本原则。Wirfs-Brock 和她的同事 [Wir90] 在给类分配职责时建议了以下 5 个指导原则。

1. 智能系统应分布在所有类中以求最大程度地满足问题的需求。每个应用系统都包含一定程度的智能，也就是系统所知道的以及所能完成的。智能在类中可以有多种分布方式。建模时可以把“不灵巧”（Dumb）类（几乎没有职责的类）作为一些“灵巧”类（有很多职责的类）的从属。尽管该方法使得系统中的控制流简单易懂，但同时有如下缺点：把所有的智能集中在少数类，使得变更更为困难；将会需要更多的类，因此需要更多的开发工作。

如果智能系统更平均地分布在应用系统的所有类中，每个对象只了解和执行某些

网络资源 关于这些类的类型的精彩讨论可以参阅 <http://www.oracle.com/technetwork/develop-tools/jdev/gettingstartedwithumlclassmodeling-130316.pdf>。

引述 对象可以科学地分为三个主要类别：不工作的、损坏的和丢失的。
Russell Baker

提问 为类分配职责时可以采用什么指导原则？

事情（通常是适度集中），并提高系统的内聚性[⊖]，这将提高软件的可维护性并减少变更的副作用。

为了确定分布智能系统是否恰当，应该评估每个 CRC 模型索引卡上标记的职责，以确定某个类是否应该具有超长的职责列表。如果有这种情况就表明智能太集中[⊖]。此外，每个类的职责应表现在同一抽象层上。例如在聚合类 `CheckingAccount` 操作列表中，评审人员注意到两项职责：账户余额和已结算的支票。第一个操作的职责意味着复杂的算术和逻辑过程，第二个操作的职责是指简单的办事员活动。既然这两个操作不是相同的抽象级别，因此已结算的支票应该被放在 `CheckEntry` 的职责中，这是由聚合类 `CheckingAccount` 压缩得到的一个类。

2. 每个职责的说明应尽可能具有普遍性。这条指导原则意味着应在类的层级结构的上层保持职责（属性和操作）的通用性（因为它们更有一般性，将适用于所有的子类）。
3. 信息和与之相关的行为应放在同一个类中。这实现了面向对象原则中的封装，数据和操作数据的处理应包装在一个内聚单元中。
4. 某个事物的信息应局限于一个类中而不要分布在多个类中。通常应由一个单独的类负责保存和操作某特定类型的信息。通常这个职责不应由多个类分担。如果信息是分布的，软件将变得更加难以维护，测试也会面临更多挑战。
5. 适合时，职责应由相关类共享。很多情况下，各种相关对象必须在同一时间展示同样的行为。例如，考虑一个视频游戏，必须显示如下类：`Player`、`PlayerBody`、`PlayerArms`、`PlayerLegs` 和 `PlayerHead`。每个类都有各自的属性（例如 `position`、`orientation`、`color` 和 `speed`），并且所有属性都必须在用户操纵游戏杆时进行更新和显示。因此，每个对象必须共享职责 `update` 和 `display`。`Player` 知道在什么时候发生了某些变化并且需要 `update` 操作。它和其他对象协作获得新的位置或方向，但是每个对象控制各自的显示。

协作。类用一种或两种方法来实现其职责：（1）类可以使用其自身的操作控制各自的属性，从而实现特定的职责；（2）类可以和其他类协作。`Wirfs-Brock` 和她的同事 [Wir90] 这样定义协作：

协作是以客户职责实现的角度表现从客户到服务器的请求。协作是客户和服务器之间契约的具体实现……如果为了实现某个职责需要发送任何消息给另一个对象，我们就说这个对象和其他对象有协作。单独的协作是单向流，即表示从客户到服务器的请求。从客户的角度看，每个协作都和服务器的某个特定职责实现相关。

要识别协作可以通过确认类本身是否能够实现自身的每个职责。如果不能实现每个职责，那么需要和其他类交互，因此就要有协作。

例如，考虑 `SafeHome` 的安全功能。作为活动流程的一部分，`ControlPanel` 对象必须确定是否启动所有的传感器，定义名为 `determine-sensor-status()` 的职责。如果传感器是开启的，那么 `ControlPanel` 必须设置属性 `status` 为“未准备好”。传感器信息可以从每个 `Sensor` 对象获取，因此只有当 `ControlPanel` 和 `Sensor` 协作时才能实现 `determine-sensor-status()` 职责。

⊖ 内聚性是一个设计概念，将在第 11 章中讨论。

⊖ 在这种情况下，可能需要将一个类分成多个类或子系统，以便更有效地分布智能。

为帮助识别协作者，分析师可以检查类之间三种不同的通用关系 [Wir90]：(1) is-part-of (是……一部分) 关系；(2) has-knowledge-of (有……的知识) 关系；(3) depends-upon (依赖……) 关系。在下面的段落中将简单地分别说明这三种通用关系。

属于某个聚合类一部分的所有类可通过 is-part-of 关系和聚合类连接。考虑前面提到的视频游戏中所定义的类，PlayerBody 是 Player 的一部分，PlayerArms、PlayerLegs 和 PlayerHead 也类似。在 UML 中，使用如图 9-4 所示的聚合方式表示这些关系。

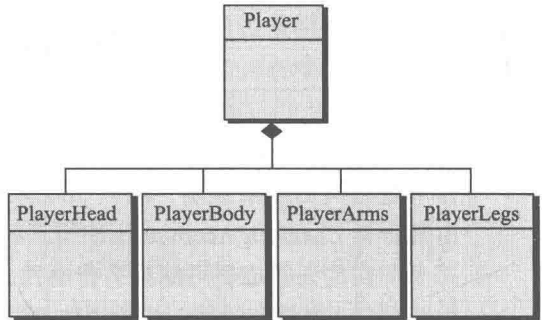


图 9-4 复合聚合类

当一个类必须从另一个类中获取信息时，就建立了 has-knowledge-of 关系。前面所说的 determine-sensor-status() 职责就是 has-knowledge-of 关系的一个例子。

depends-upon 关系意味着两个类之间具有 has-knowledge-of 和 is-part-of 不能实现的依赖关系。例如，PlayerHead 通常必须连接到 PlayerBody（除非视频游戏特别暴烈），然而每个对象并没有其他对象的直接信息。PlayerHead 对象的 center-position 属性由 PlayerBody 的中心位置确定，该信息通过第三方对象 Player 获得，即 PlayerBody 需要 Player。因此，PlayerHead 依赖 PlayerBody。

在所有情况下，我们都把协作类的名称记录在 CRC 模型索引卡上，紧靠在协作的职责旁边。因此，索引卡包含一个职责列表以及相关的能够实现这些职责的协作（图 9-3）。

当开发出一个完整的 CRC 模型时，利益相关者可以使用如下方法评审模型 [Amb95]。

1. 所有参加（CRC 模型）评审的人员拿到一部分 CRC 模型索引卡。拆分协作卡片（也就是说每个评审员不得有两张存在协作关系的卡片）。
2. 分类管理所有的用例场景（以及相关的用例图）。
3. 评审组长细致地阅读用例。当评审组长看到一个已命名的对象时，给拥有相应类索引卡的人员一个令牌。例如，SafeHome 的一个用例包含如下描述：

房主观察 SafeHome 控制面板以确定系统是否已经准备接收输入。如果系统没有准备好，房主必须手工关闭窗口（门）以便指示器呈现就绪状态。（未就绪指示器意味着某个传感器是开启的，也就是说某个门或窗户是打开的。）

当评审组长看到用例说明中的“控制面板”，就把令牌传给拥有 ControlPanel 索引卡的人员。“意味着某个传感器是开启的”语句需要索引卡包含确认该含义的职责（由 determine-sensor-status() 实现该职责）。靠近索引卡职责的是协作者 Sensor，然后令牌传给 Sensor 对象。

4. 当令牌传递时，该类卡的拥有者需要描述卡上记录的职责。评审组确定（一个或多个）职责是否满足用例需求。
5. 如果记录在索引卡上的职责和协作不能满足用例，就需要修改卡片。修改可能包括定义新类（和相关的 CRC 索引卡），或者在已有的卡上说明新的或修改的职责、协作。

该过程持续进行直到用例（或用例图）编写结束。评审完所有用例后将继续进行需求建模。

SafeHome CRC 模型

[场景] Ed 的办公室，刚开始需求建模。

[人物] Vinod 和 Ed，SafeHome 软件工程项目团队成员。

[对话]

(Vinod 已经决定通过一个例子向 Ed 展示如何开发 CRC 卡。)

Vinod：在你着手于监视功能而 Jamie 忙着安全功能的时候，我在准备住宅管理功能。

Ed：情况怎样？市场营销人员的想法总是在变化。

Vinod：这是整个功能的第一版用例……我们已经改进了一点，它应该能提供一个整体视图。

用例：SafeHome 住宅管理功能。

说明：我们希望通过个人计算机上的住宅管理接口或互联网连接，来控制有无线接口控制器的电子设备。系统应该允许我打开或关闭指定的灯，控制连接到无线接口的设备，设置取暖和空调系统达到预定温度。为此，我希望从房屋平面图上选择设备。每个设备必须在平面图上标识出来。作为可选的特性，我希望控制所有的视听设备——音响设备、电视、DVD、数字录音机等。

通过不同选项就能够针对各种情况设置整个房屋，第一个选项是“在家”，第二个是“不在家”，第三个是“彻夜不归”，第四个是“长期外出”。所有这些情况都适用于所有设备的设置。在彻夜不归和长期外出时，系统将以随机的间隔时间开灯和关灯（以造成有人在家的错觉），并控制取暖和空调系统。我应能够通过有适当密码保护的互联网撤销这些设置……

Ed：那些负责硬件的伙计已经设计出所有的无线接口了吗？

Vinod（微笑）：他们正在忙这个，据说没有问题。不管怎样，我从住宅管理中提取

了一批类，我们可以用一个做例子。就以 HomeManagementInterface 类为例吧！

Ed：好，那么职责是什么？类及协作者的属性和操作是那些职责所指向的类。

Vinod：我想你还不了解 CRC。

Ed：可能有点，但还是继续吧。

Vinod：这就是我给出的 HomeManagementInterface 的类定义。

属性：

optionsPanel——在按钮上提供信息，用户可以使用这些信息选择功能。

situationPanel——在按钮上提供信息，用户可以使用这些信息选择环境。

floorPlan——类似于监视对象，但这个用来显示设备。

deviceIcons——图标上的信息，代表灯、家用电器、HVAC 等。

devicePanels——模拟家用电器或设备控制面板，允许控制。

操作：

displayControl(), selectControl(), displaySituation(), selectSituation(), accessFloorplan(), selectDeviceIcon(), displayDevicePanel(), accessDevicePanel()……

类：HomeManagementInterface

职责	协作者
displayControl ()	OptionsPanel (类)
selectControl ()	OptionsPanel (类)
displaySituation ()	SituationPanel (类)
selectSituation ()	SituationPanel (类)
accessFloorplan ()	FloorPlan (类)
……	……

Ed：这样，当调用 accessFloorPlan() 操作时，将和 FloorPlan 对象协作，类似我们为监视开发的对象。等一下，我这里有它的说明（他们查看图 9-2）。

Vinod：确实如此。如果我们希望评审整个类模型，可以从这个索引卡开始，然后

到协作者的索引卡，再到协作者的协作者的索引卡，依此类推。

Ed: 这真是个发现遗漏和错误的好方法。
Vinod: 的确如此。

9.5 关联和依赖

在很多例子中，两个分析类以某种方式相互联系着。在 UML 中，这些联系被称作关联（association）。参考图 9-2，通过识别 FloorPlan 与另外两个类 Camera 和 Wall 之间的一组关联确定 FloorPlan 类。类 Wall 和三个构成墙类 WallSegment、Window 和 Door 相关联。

在某些情况下，关联可以更进一步地指出多样性。参考图 9-2，一个 Wall 对象可以由一个或多个 WallSegment 对象构成。此外，Wall 对象可能包含 0 或多个 Window 对象以及 0 或多个 Door 对象。这些多样性限制如图 9-5 所示，其中“一个或多个”使用 1..* 表示，“0 或多个”使用 0..* 表示。在 UML 中，星号表示范围无上界。^①

在很多事例中，两个分析类之间存在客户-服务器关系。这种情况下，客户类以某种方式依赖于服务器类并且建立了依赖关系。依赖性是由一个构造型（stereotype）定义的。在 UML 中，构造型是一个“可扩展机制”[Arl02]，允许软件工程师定义特殊的建模元素，这些建模元素的语义是由用户自定义的。在 UML 中，构造型由一对尖括号表示（如 <<stereotype>>）。

下面举例说明 SafeHome 监视系统中的简单依赖关系。Camera 对象（本例中的服务器类）向 DisplayWindow 对象（本例中的客户类）提供视频图像。这两个对象之间的关系不是简单的关联，而是存在着依赖关系。在监视用例中（没有列出来），建模者知道必须提供特定的密码才能查看指定摄像机的位置。其实现的一种方法是让 Camera 请求密码，然后在获得 DisplayWindow 的允许后显示视频。这可以由图 9-6 表示，其中 <<access>> 意味着通过特定的密码控制对摄像机输出的使用。

关键点 关联定义了类之间的联系，多样性定义了一个类和另一个类之间联系的数量关系。

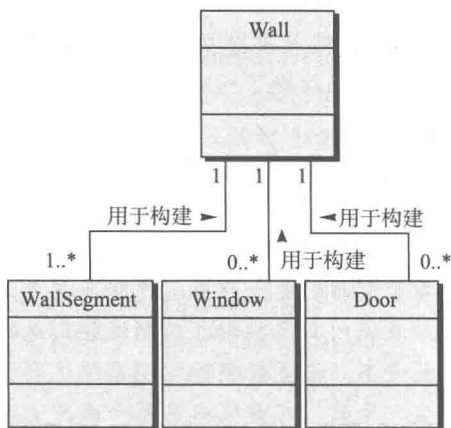


图 9-5 多样性

提问 什么是构造型？



图 9-6 依赖性

9.6 分析包

分析建模的一部分重要工作是分类，也就是将分析模型的各种元素（如用例、分析类）以一种方式分类，分组打包后称为分析包，并取一个有代表性的名字。

关键点 分析包用来集合一组相关的类。

① 其他的多样性关联（一对一、一对多、多对多、一对某指定上下限的范围，以及其他）可以标识为关联的一部分。

为了说明分析包的使用，考虑我们前面所说的视频游戏。假设视频游戏的分析模型已经建成，并且已经生成大量的类。有一些类关注游戏环境，即用户游戏时所看到的可视场景。Tree、Landscape、Road、Wall、Bridge、Building、VisualEffect 类等可能就属于游戏环境类。另一些类关注游戏内的人物，说明他们的肢体特点、动作和局限。也可能定义了（前面提到的）Player、Protagonist、Antagonist、SupportingRoles 类。还需要一些类用来说明游戏的规则，即游戏者如何穿越环境。例如这里可以选 RulesOfMovement 类和 ConstraintsOnAction 类。还可能存在很多其他类，可以用图 9-7 中的分析包表示这些类。

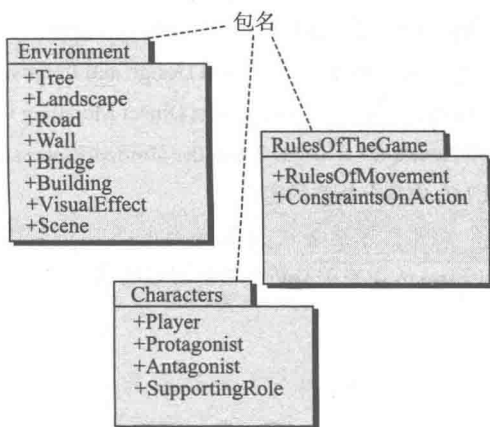


图 9-7 包

每个包中分析类名字前的加号表示该类是公共可见的，因此可以从其他包访问。尽管目前的图中没有显示，但包中的任何元素之前都可以添加其他符号。负号表示该元素对其他包是隐藏的，# 号表示只能由指定包中的类访问该元素。

习题与思考题

9.1 构建如下系统中的一个：

- 你所在大学中基于网络的课程注册系统。
- 一个计算机商店的基于 Web 的订单处理系统。
- 一个小企业的简单发票系统。
- 内置于电磁灶或微波炉的互联网食谱。

选择你感兴趣的系统开发一套处理说明。然后使用语法解析技术识别候选对象和类。

9.2 使用问题 9.1 中识别的类开发一系列操作。

9.3 为问题 8.5 中表述的 PHTRS 系统开发一个类模型。

9.4 为 9.4 节中非正式描述的 SafeHome 住宅管理系统编写一个基于模板的用例。

9.5 为问题 9.1 所选的产品或系统开发一组完整的 CRC 模型索引卡。

9.6 指导你的同事一起评审 CRC 索引卡，评审结果中增加了多少类、职责和协作者？

9.7 什么是分析包？如何使用分析包？

扩展阅读与信息资源

Weisfeld 讨论了常规的基于类的观点（《The Object-Oriented Thought Process》，4th ed., Addison-Wesley, 2013）。讨论基本类的建模方法的书包括：Bennet 和 Farmer（《Object-Oriented Systems Analysis and Design Using UML》，McGraw-Hill, 2010），Ashrafi（《Object-Oriented Systems Analysis and Design》，Prentice Hall, 2008），Booch（《Object-Oriented Analysis and Design with Applications》，3rd ed., Addison-Wesley, 2007），George 和他的同事（《Object-Oriented Systems Analysis and Design》，2nd ed., Prentice Hall, 2006），O'Docherty（《Object-Oriented Analysis and Design》，Wiley, 2005），Satzinger 等人（《Object-Oriented Analysis and Design with the Unified Process》，Course Technology, 2004），Stumpf 和 Teague（《Object-Oriented Systems Analysis and Design with UML》，Prentice Hall, 2004）。

UML 建模技术可以应用于分析和设计，讨论这方面的书包括：Dennis 和他的同事（《Systems

Analysis and Design with UML Version 2.0》, Wiley, 4th ed., 2012), Ramnath 和 Dathan (《Object-Oriented Analysis and Design》, Springer, 2011), Bennett 和 Farmer (《Object-Oriented System Analysis and Design Using UML》, McGraw-Hill, 4th ed., 2010), Larman (《Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development》, Dohrling Kindersley, 2008), Rosenberg 和 Stephens (《Use Case Driven Object Modeling with UML Theory and Practice》, Apress, 2007), 还有 Arlow 和 Neustadt (《UML 2 and the Unified Process》, 2nd ed., Addison-Wesley, 2005)。这些书都描述了用 UML 内容分析类的定义。

网上有很多关于基于类的方法进行需求建模的信息。可以参考 SEPA 网站 www.mhhe.com/pressman 上有关分析建模的最新参考文献。

需求建模：行为和模式

要点浏览

概念：在这一章将学习需求建模的其他维度——面向行为建模、模式，以及开发 WebApp 时要考虑的特定需求分析问题。这里的每种模型都是第 8 章和第 9 章的补充。

人员：软件工程师（有时称为分析师）从各个利益相关者中提取需求进行建模。

重要性：软件工程师洞悉到软件需求的增长与来自不同需求模型维度的增长成正比。尽管可能没有时间，缺乏资源，无法采用第 8～10 章所建议的任何一种表示方法进行开发，但是软件工程师有必要认知每种不同的建模方式，这会给他提供审视问题的不同方法。继而他（和其他利益相关者）将能够更好地进入状态，

更好地界定是否已把必须完成的需求真正描述清楚。

步骤：行为建模描述了系统及其类的状态，以及事件对这些类的影响。基于模式的建模利用现有领域的知识使得需求分析更为容易。WebApp 的需求模型特别适用于表述内容、交互操作、功能和配置相关的需求。

工作产品：可为需求建模选择大量不同的基于文本和图形的表示方法。每种表示方法都提供了一种或多种模型元素。

质量保证措施：必须评审需求建模产品的正确性、完备性和一致性。必须反映所有利益相关者的要求，为导出设计建立基础。

在第 8 章和第 9 章讨论了基于场景建模和基于类模型后，我们很自然会问：“这些需求建模表示方法就足够了吗？”

唯一合理的回答是：“要看情况而定。”

对于某种类型的软件，用例可能是唯一可行的需求建模表示方法。而对于其他类型的软件，则需要选择面向对象的方法开发基于类的模型。但在另外一些情形下，可能必须要对复杂应用软件需求做个检测：查看当数据对象在系统中移动时是如何转换的；查看作为外部事件后果的应用软件是如何工作的；查看现存知识领域能否解决当前问题；或者在基于 Web 的系统或移动系统和应用软件中，如何将内容和功能融合在一起，使得最终用户能够利用 WebApp 实现相关目标。

关键概念

分析模式
行为模型
事件
顺序图
状态图
状态表达

10.1 生成行为模型

前面章节讨论的建模表示方法表达了需求模型中的静态元素，现在则是把它转换成系统或产品的动态行为的时候了。要实现这一任务，可以把系统行为表示成一个特定的事件和时间的函数。

行为模型显示了软件如何对外部事件或激励做出响应。要生成模型, 分析师必须按如下步骤进行: (1) 评估所有的用例, 以保证完全理解系统内的交互顺序; (2) 识别驱动交互顺序的事件, 并理解这些事件如何与特定的对象相互关联; (3) 为每个用例生成序列; (4) 创建系统状态图; (5) 评审行为模型以验证准确性和一致性。

在下面的几节中将讨论每个步骤。

10.2 识别用例事件

在第 8 章中, 我们知道用例表现了涉及参与者和系统的活动顺序。一般而言, 只要系统和参与者之间交换了信息就发生事件。事件应该不是被交换的信息, 而是已交换信息的事实。

为了说明如何从信息交换的角度检查用例, 让我们再来考察 SafeHome 安全功能中的一部分用例。

房主使用键盘键入 4 位密码。系统将该密码与已保存的有效密码相比较。如果密码不正确, 控制面板将鸣叫一声并复位以等待下一次输入; 如果密码正确, 控制面板等待进一步的操作。

用例场景中加下划线的部分表示事件。应确认每个事件的参与者, 应标记交换的所有信息, 而且应列出任何条件或限制。

举个典型事件的例子, 考虑用例中加下划线的“房主使用键盘键入 4 位密码”。在需求模型的环境下, 对象 Homeowner^①向对象 ControlPanel 发送一个事件。这个事件可以称作输入密码。传输的信息是组成密码的 4 位数字, 但这不是行为模型的本质部分。重要的是注意到某些事件对用例的控制流有明显影响, 而其他事件对控制流没有直接影响。例如, 事件输入密码不会明显地改变用例的控制流, 但是事件比较密码(从与事件“系统将该密码与保存的有效密码相比较”的交互中得到)的结果将明显影响到 SafeHome 软件的信息流和控制流。

一旦确定了所有的事件, 这些事件将被分配到所涉及的对象, 对象负责生成事件(例如, Homeowner 房主生成输入密码事件)或识别已经在其他地方发生的事件(例如, ControlPanel 控制面板识别比较密码事件的二元结果)。

10.3 状态表达

在行为建模中, 必须考虑两种不同的状态描述: (1) 系统执行其功能时每个类的状态; (2) 系统执行其功能时从外部观察到的系统状态。

类状态具有被动和主动两种特征 [Cha93]。被动状态只是某个对象所有属性的当前状态。例如, 类 Player 的被动状态(在第 9 章讨论的视频游戏应用程序中)将包含 Player 当前的 position 和 orientation 属性, 以及和游戏相关的 Player 的其他特性(例如, 用来显示 magic wishes remaining 的属性)。对象的主动状态指的是对象进行持续变换或处理时的当前状态。类 Player 可能具有如下的主动状态: 移动、休息、受伤、疗伤、被捕、失踪等。事件(有时称为触发器)才能迫使对象做出从一个主动状态到另一个主动状态的转移。

提问 如何用模型表明软件对某些外部事件的影响?

关键点 系统状态可以表现特定的外部可观察的行为, 类的状态可以表现当前系统执行功能时的行为。

① 在这个例子中, 我们假设每位与 SafeHome 交互的用户(房主)都拥有确认的密码, 因此都是合法的对象。

下面将讨论两种不同的行为表现形式。第一种显示一个类如何改变基于外部事件的状态，第二种以时间函数的形式显示软件的行为。

分析类的状态图。UML 状态图^①就是一种行为模型，该图为每个类呈现了主动状态和导致这些主动状态发生变化的事件（触发器）。图 10-1 举例说明了 SafeHome 安全功能中 ControlPanel 类的状态图。

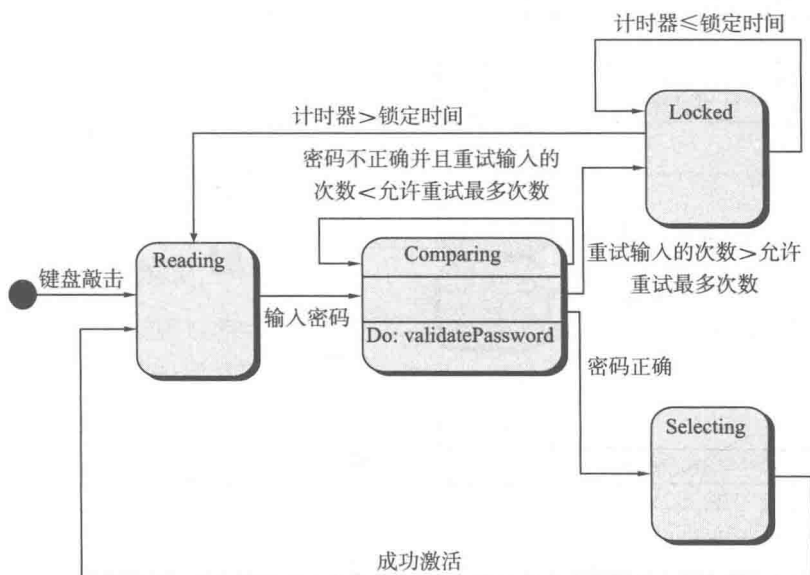


图 10-1 ControlPanel 类的状态图

图 10-1 中显示的每个箭头表示某个对象从一个主动状态转移到另一个主动状态。每个箭头上的标注都体现了触发状态转移的事件。尽管主动状态模型在提供对象的“生命历史”信息方面非常有用，但也能提供另外一些信息以便更深入地理解对象的行为。除了说明导致转移发生的事件外，分析师还可以说明守卫和动作 [Cha93]。守卫是为了保证转移发生而必须满足的一个布尔条件。例如，图 10-1 中从“读取”状态转移到“比较”状态的守卫可以由检查用例来确定：

```
if (password input = 4 digits) then compare to stored password
```

一般而言，转移的守卫通常依赖于某个对象的一个或多个属性值。换句话说，守卫依赖于对象的被动状态。

动作是与状态转移同时发生的，或者作为状态转移的结果，通常包含对象的一个或多个操作（职责）。例如，和输入密码事件（见图 10-1）相关联的动作是由 validatePassword() 操作的，该操作访问 password 对象并通过执行按位比较来验证输入的密码。

顺序图。第二种表现行为的方式在 UML 中称作顺序图，它表明事件如何引发从一个对象到另一个对象的转移。一旦通过检查用例确认了事件，建模人员就创建了一个顺序图，即用时间函数表现如何引发事件从一个对象流到另一个对象。事实上，顺序图是用例的速记版本。它表现了导致行为从一个类流

关键点 与不注明相关类表现行为的状态图不同，顺序图通过说明类如何从一个状态转移到另一状态来表现行为。

① 如果读者对 UML 不熟悉，在附录 1 中有这些重要建模符号的简单介绍。

到另一个类的关键类和事件。

图 10-2 给出了 SafeHome 安全功能的部分顺序图。每个箭头代表了一个事件（源自一个用例）并说明了事件如何引导 SafeHome 对象之间的行为。时间纵向（向下）度量，窄的纵向矩形表示处理某个活动所用的时间。沿着纵向的时间线可以展示出对象的状态。

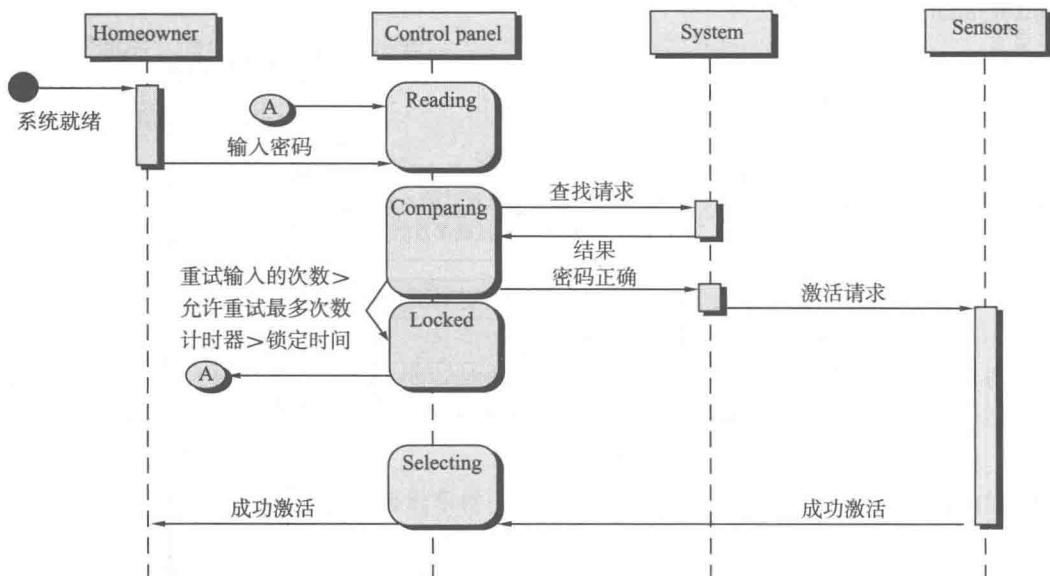


图 10-2 SafeHome 安全功能的顺序图（部分）

第一个事件系统就绪来自于外部环境并且把行为引导到对象 Homeowner。房主输入一个密码，查找请求事件发送到 System，它在一个简单的数据库中查找密码并向 ControlPanel（现在处在比较状态）返回（找到或没有找到）结果。有效的密码将促使系统产生密码正确事件，该事件通过激活请求事件激活传感器。最终，通过控制把成功激活事件返回到房主。

一旦完成了完整的顺序图，就把所有导致系统对象之间转移的事件整理为输入事件集合和输出事件集合（来自一个对象）。对于将要构建的系统而言，这些信息对于创建系统的有效设计非常有用。

软件工具 UML 中的通用分析建模

【目标】分析建模工具可以使用 UML 语言开发基于场景的模型、基于类的模型和行为模型。

【机制】这类工具支持构建分析模型所需的全部 UML 图（这些工具也支持设计建模）。除了制图，这类工具：（1）为所有的 UML 图执行一致性和正确性检查；

（2）为设计和代码生成提供链接；（3）构建数据库，以便实现管理和评估复杂系统所需的大型 UML 模型。

【代表性工具】^①

如下工具支持分析建模所需的全部 UML 图：

- ArgoUML。开源工具（argouml.tigris。

① 这里记录的工具只是此类工具的例子，并不代表本书支持采用这些工具。大多数情况下，工具的名字由各自的开发者注册为商标。

- org)。
- Enterprise Architect。由 Sparx Systems 开发 (www.sparxsystems.com.au)。
- PowerDesigner。由 Sybase 开发 (www.sybase.com)。
- Rational Rose。由 IBM (Rational) 开发 (www-01.ibm.com/software/rational)。
- Rational System Architect。由 Popkin Software 开发, 现在归属于 IBM (<http://www-01.ibm.com/software/awdtools/systemarchitect/>)。
- UML Studio。由 Pragsoft Corporation 开发 (www.pragsoft.com)。
- Visio。由 Microsoft 开发 (<http://office.microsoft.com/en-gb/visio/>)。
- Visual UML。由 Visual Object Modelers 开发 (www.visualuml.com)。

10.4 需求建模的模式

软件模式是获取领域知识的一种机制, 在遇到新问题时可以反复使用。在某些情况下, 领域知识在同一应用领域中用于解决新问题。在另外一些情况下, 通过模式获取的领域知识可借助模拟用于完全不同的应用领域。

分析模式的最初创作者没有“创建”模式, 但在需求工程工作中发现了模式。一旦发现模式则记载“明确的常见问题: 哪种模式适用, 规定的解决方案, 在实践中使用模式的假设和约束, 以及关于模式的某些常见的其他消息, 如使用模式的动机和驱动力, 讨论模式的优缺点, 参考在实践应用中某些已知的使用模式的样例” [Dev01]。

第7章引入了分析模式的概念并指明这些模式表示了某些应用领域(例如一个类、一个功能或一个行为), 当为这个领域的应用执行需求建模时会重复使用这些模式。分析模式都存储在一个仓库中, 以便软件团队的成员能够使用搜索工具找到并复用。一旦选择到合适的模式, 就可以通过引用模式名称将其整合到需求模型中。

10.4.1 发现分析模式

需求模型由各种元素组成: 基于场景(用例)、基于类(对象和类)和行为(事件和状态)。其中每个元素都是从不同的视角检查问题, 并且每一个都提供一种发现模式的机会, 可能发生在整个应用领域, 或者发生在类似但不同的应用领域。

在需求模型的最基本的元素是用例。在这个讨论环境下, 一套连贯用例可以成为服务于发现一个或多个分析模式的基础。语义分析模式 (Semantic Analysis Pattern, SAP) “描述了一小套连贯用例, 这些用例一起描述了通用应用的基础” [Fer00]。

下面我们考虑要求控制和监控“实时摄像机”以及汽车临近传感器的软件用例。

用例: 监控反向运动。

描述: 当车辆安装了反向装置后, 控制软件就能将后向摄像机的视频输入仪表板显示器。控制软件在仪表板显示器上叠加各种距离线和方向线, 以便车辆向后运动时驾驶员能保持方向。控制软件还能监控临近传感器, 以判定在车后方 10 英尺内是否有物体存在。如果临近传感器检测到某个物体在车后方 x 英尺内就会让车自动停止, 这个 x 值由车辆的速度决定。

在需求收集和建模阶段, 本用例包含(在一套连贯用例中)各种将要精炼和详细阐述的功能。无论完成得如何精炼, 建议用例要简单, 但还要广泛地适用于 SAP, 即具有基于软件

的监控和在一个物理系统中对传感器和执行器的控制。在本例中，“传感器”提供临近信息和视频信息。“执行器”用于车辆的停止系统（如果一个物体离车辆很近就会调用它）。但是更常见的情况是发现大量的应用模式。

许多不同应用领域的软件需要监控传感器和控制物理执行器。所依照的分析模式描述了能广泛应用的通用需求。这个模式叫做 Actuator-Sensor（执行器-传感器），将用作 SafeHome 的部分需求模型，将在随后的 10.4.2 节讨论。

SafeHome | 发现分析模式

[场景] 一个会议室，一个团队会议。

[人物] Jamie Lazar，软件团队成员；Ed Robbins，软件团队成员；Doug Miller，软件工程经理。

[对话]

Doug: SafeHome 项目有关传感器网络的需求建模进展如何啊？

Jamie: 对我来说传感器的工作有点新，但我认为我能掌控好。

Doug: 我们能帮上你什么忙吗？

Jamie: 如果我曾建立过类似的系统，会更容易些。

Doug: 当然。

Ed: 我考虑到在这种情况下我们要找到一个分析模式，帮助我们为这些需求建模。

Doug: 如果我们能发现正确的模式，我们可以避免重新发明轮子。

Jamie: 对我而言这太好了。那么我们怎么开始呢？

Ed: 我们有一个库包含大量的分析和设计模式。我们只需要检索出符合需求的模式。

Doug: 看来就这么干吧。你认为怎样，Jamie？

Jamie: 如果 Ed 能帮我开始，我今天就按此行动了。

10.4.2 需求模式举例：执行器-传感器[⊖]

SafeHome 安全功能需求之一是能监控安全传感器（例如，入侵传感器，火、烟或一氧化碳传感器，水传感器）。基于互联网的 SafeHome 还将要求能控制住所内安全摄像机的活动（例如摇摄、变焦）。这意味着 SafeHome 软件必须管理各种传感器和“执行器”（例如摄像机控制机制）。

Konrad 和 Cheng[Kon02] 建议将需求模式命名为执行器-传感器，它为 SafeHome 软件这项需求的建模提供有用的指导。执行器-传感器模式的简约版本最初是为如下的汽车应用开发的。

模式名：执行器-传感器。

目的：详细说明在嵌入系统中的各种传感器和执行器。

动机：嵌入系统包含各种传感器和执行器，这些传感器和执行器都直接或间接连接到一个控制单元。虽然许多传感器和执行器看上去完全不同，但它们的行为是相似的，足以构成一个模式。这种模式显示了如何为系统指定传感器和执行器，包括属性和操作。执行器-传感器模式为被动式传感器使用拉机制（对消息的显示要求），为主动传感器使用推机制（消息广播）。

[⊖] 本节采用 [Kon02] 的内容，已得到作者许可。

约束：

- 每个被动传感器必须通过某种方法来读取传感器的输入和表示传感器值的属性。
- 每个主动传感器必须能在其值发生变更时广播更新消息。
- 每个主动传感器应该能发送一个生命刻度，即在特定时间帧中发布状态信息，以便检测出错误动作。
- 每个执行者必须通过某种方法来调用由 ComputingComponent 计算构件决定的适当应答。
- 每个传感器和执行器应有实施检测其自身操作状态的功能。
- 每个传感器和执行器应能测试接收值或发送值的有效性，并且当值超出指定边界时能设定其操作状态。

适用性：对有多个传感器和执行器的任何系统都是非常有用的。

结构体：图 10-3 显示了执行器 - 传感器模式的 UML 类图，执行器、被动传感器和主动传感器是抽象类。这个模式中有四种不同的传感器和执行器。

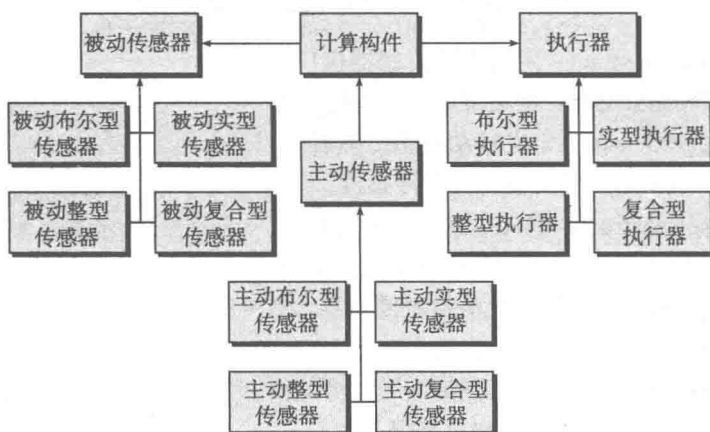


图 10-3 传感器和执行器的 UML 类图

传感器和执行器的常见类型为 Boolean、Integer 和 Real。就基本数据类型而言，复杂类是不能用值简单表示的，例如雷达设备。虽然如此，这些设备还是应该继承来自抽象类的接口，因为它们应该具备基本的功能（如查询操作状态）。

行为：图 10-4 描述了执行器 - 传感器例子的 UML 顺序图。这个例子可以应用于 SafeHome 功能，用以控制安全摄像机的调整（例如摇摄、变焦）。在读取或设置值时，ControlPanel 需要一个传感器（被动传感器）和一个执行器（摇动控制器）来进行以诊断为目的的操作状态核查。名为 Set Physical Value 和 Get Physical Value 的消息不是对象间的信息，相反，它们描述了系统物理设备和相关软件对应项之间的交互活动。在图的下部（即在水平线以下），PositionSensor 报告操作状态为零。接着 ComputingComponent 发送位置传感器失败的错误代码给 FaultHandler 错误处理程序，它将决定这些错误对系统的影响，以及需要采取的措施。从传感器获得的数据经过计算得到执行器所需的应答。

参与者：模式“详细列举了包含在需求模式中的类或对象”[Kon02]，并描述了每个类或对象（图 10-3）的职责。简单列表如下：

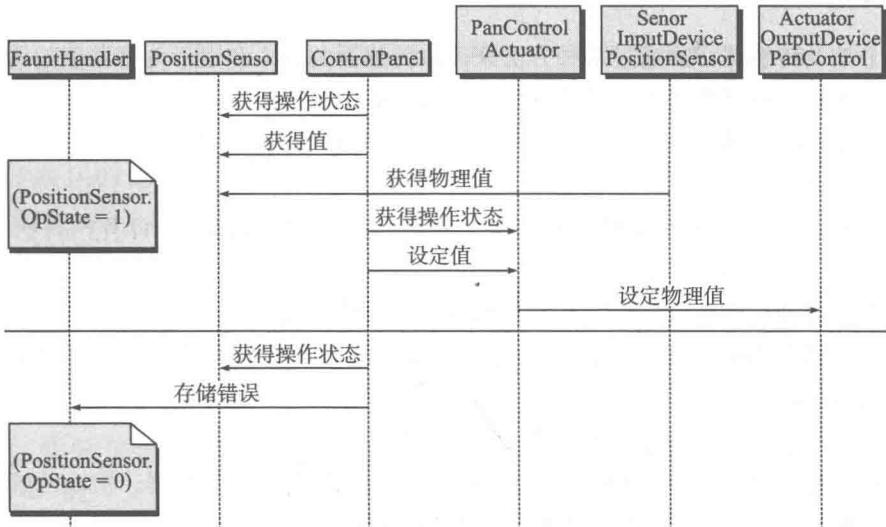


图 10-4 执行器-传感器模式的 UML 顺序图

- PassiveSensor abstract: 定义被动传感器接口。
- PassiveBooleanSensor: 定义被动布尔型传感器。
- PassiveIntegerSensor: 定义被动整型传感器。
- PassiveRealSensor: 定义被动实型传感器。
- ActiveSensor abstract: 定义主动传感器接口。
- ActiveBooleanSensor: 定义主动布尔型传感器。
- ActiveIntegerSensor: 定义主动整型传感器。
- ActiveRealSensor: 定义主动实型传感器。
- Actuator abstract: 定义执行器接口。
- BooleanActuator: 定义布尔型执行器。
- IntegerActuator: 定义整型执行器。
- RealActuator: 定义实型执行器。
- ComputingComponent: 控制器的中心部分, 它从传感器得到数据并为执行器计算所需的应答。
- ActiveComplexSensor: 主动复合型传感器具备抽象类 ActiveSensor 的基本功能, 但还需要指定额外更详细的方法和属性。
- PassiveComplexSensor: 被动复合型传感器具备抽象类 PassiveSensor 的基本功能, 但还需要指定额外更详细的方法和属性。
- ComplexActuator: 复合型执行器具备抽象类 Actuator 的基本功能, 但还需要指定额外更详细的方法和属性。

协作。本节描述的是对象和类之间如何进行交互活动, 以及它们各自如何实现自身的责任。

- 当 ComputingComponent 需要更新 PassiveSensor 的值时, 它询问传感器, 通过发送适当的消息请求传值。
- ActiveSensor 无需查询。这些对象和类启动传送过程, 向计算机单元中传送传感器的

值，使用适当方法设定在 ComputingComponent 中的值。对象和类在指定的时间帧发送至少一次生命刻度，以使用系统的时钟时间更新它们的时间戳。

- 当 ComputingComponent 需要设定执行器的值时，便会给执行器发送值。
- ComputingComponent 能使用适当的方法查询和设定传感器和执行器的操作状态。如果发现操作状态为零，就发送错误给 FaultHandler 错误处理程序，这个类包含处理错误消息的方法，比如启动更详细的恢复机制或者一个备份设备。如果不可恢复，系统只能使用传感器最后的已知值或者默认值。
- ActiveSensor 提供增加或消除地址或者组件地址范围的方法，一旦值发生变化，组件就能获得消息。

结果：

1. 传感器和执行器类有一个通用接口。
2. 只能通过消息访问类的属性，并且类决定是否接受这个消息。例如，如果执行器的值超过其最大值，执行器类就不可能接收消息，或者使用默认的最大值。
3. 系统潜在的复杂度得以降低是因为传感器和执行器有统一的接口。

需求模式的描述也能对其他相关的需求模式和设计模式提供参考。

习题与思考题

- 10.1 表示行为建模时有两种不同“状态”类型，它们是什么？
- 10.2 如何从状态图区分顺序图？它们有何相似之处？
- 10.3 为现代移动手机建议 3 种需求模式，并简要描述每种模式。这些模式能否被其他设备使用？举例说明。
- 10.4 在问题 10.3 中选择一个你要开发的模式，模拟 10.4.2 节中表达的某个内容和模型，开发出一个合理并完整的模式描述。
- 10.5 你认为 www.safehomeassured.com 需要多少种分析模型？
- 10.6 WebApp 交互建模的目的是什么？
- 10.7 引起争议的问题是 WebApp 的功能模型应延期开发直至设计阶段。表述关于这个议题的支持和反对观点。
- 10.8 配置模型的目的是什么？
- 10.9 如何从交互模型中区分导航模型？

扩展阅读与信息资源

行为建模呈现了系统行为的动态视图。Samek (《Practical UML Statecharts in C/C++: Event Driven Programming for Embedded Systems》, CRC Press, 2008)、Wagner 和他的同事 (《Modeling Software with Finite State Machines: A Practical Approach》, Auerbach, 2006) 以及 Boerger 和 Staerk (《Abstract State Machines》, Springer, 2003) 深入讨论了状态图和其他行为的表示方法。Gomes 和 Fernandez (《Behavioral Modeling for Embedded Systems and Technologies》, Information Science Reference, 2009) 为描述嵌入式系统的行为模型编辑了一本选集。

为软件模式所写的大部分书都关注软件设计。而 Vaughn (《Implementing Domain-Driven Design》, Addison-Wesley, 2013)、Whithall (《Software Requirement Patterns》, Microsoft Press, 2007)、Evans (《Domain-Driven Design》, Addison-Wesley, 2003) 和 Fowler ([Fow03], [Fow97]) 的书都是专门写分析模式的。

Pressman 和 Lowe[Pre08] 表述了需要深层次讨论的 WebApp 分析模型。Rossi 和他的同事 (《 Web Engineering: Modeling and Implementing Web Applications 》, Springer, 2010) 以及 Neil (《 Mobile Design Pattern Gallery: UI Patterns 》, O'Reilly, 2012) 讨论在应用开发中模式的使用方法。Murugesan 和 Desphande 编写的论文集中包含了一篇文章 (《 Web Engineering: Managing Diversity and Complexity of Web Application Development 》, Springer, 2001), 处理了 WebApp 需求的各个方面。另外《 International Conference on Web Engineering 》论文集每年都会发表解决需求建模问题的文章。

网上有大量需求建模方面的资源。关于分析建模的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 上找到。

设计概念

要点浏览

概念: 几乎每位工程师都希望做设计工作。设计体现了创造性——利益相关者的需求、业务要求和技术考虑都集中地体现在产品或系统的形成中。设计创建了软件的表示或模型, 但与需求模型(注重描述所需的数据、功能和行为)不同, 设计模型提供了软件体系结构、数据结构、接口和构件的细节, 而这些都是实现系统所必需的。

人员: 软件工程师负责处理每一项设计任务。

重要性: 设计是让软件工程师为将要构建的系统或产品建立模型。在生成代码、进行测试以及大量最终用户使用之前, 要对模型的质量进行评估, 并进行改进。软件质量是在设计中建立的。

步骤: 设计可以采用很多不同的方式描述

软件。首先, 必须表示系统或产品的体系结构; 其次, 为各类接口建模, 这些接口在软件和最终用户、软件和其他系统与设备以及软件和自身组成的构件之间起到连接作用; 最后, 设计构成系统的软件构件。每个视图表现了不同的设计活动, 但所有的视图都要遵循一组指导软件设计工作的基本设计概念。

工作产品: 在软件设计过程中, 包含体系结构、接口、构件级和部署表示的设计模型是主要的工作产品。

质量保证措施: 软件团队从以下各方面来评估设计模型: 确定设计模型是否存在错误、不一致或遗漏; 是否存在更好的可选方案; 设计模型是否可以在已经设定的约束、时间进度和成本内实现。

软件设计包括一系列原理、概念和实践, 可以指导高质量的系统或产品开发。设计原理建立了指导设计工作的最重要原则。在运用设计实践的技术和方法之前, 必须先理解设计概念, 设计实践本身会产生软件的各种表示, 以指导随后的构建活动。

设计是软件工程是否成功的关键。在 20 世纪 90 年代早期, Lotus 1-2-3 的创始人 Mitch Kapor 在《Dr. Dobbs》杂志上发表了如下“软件设计宣言”:

什么是设计? 设计是你身处两个世界——技术世界和人类的目标世界, 而你尝试将这两个世界结合在一起……

罗马建筑批评家 Vitruvius 提出了这样一个观念: 设计良好的建筑应该展示出坚固、适用和令人愉悦的特点。对好的软件来说也是如此。坚固: 程序应该不含任何妨碍其功能的缺陷。适用: 程序应该符合开发的目标。愉悦: 使用程序的体验应是愉快的。本章, 我们开始介绍软件设计理论。

关键概念

- 抽象
- 体系结构
- 方面
- 内聚
- 数据设计
- 设计过程
- 功能独立
- 良好的设计
- 信息隐蔽
- 模块化
- 面向对象的设计
- 模式
- 质量属性
- 质量指导原则

关键概念

重构
关注点分离
软件设计
逐步求精

引述 软件工程

最常见的奇迹是从分析到设计以及从设计到代码的转换。

Richard Due^①

设计的目标是创作出坚固、适用和令人愉悦的模型或表示。为此，设计师必须先实现多样化，然后再进行聚合。Belady[Bel81]指出，“多样化是指要获取所有方案和设计的原始资料，包括目录、教科书和头脑中的构件、构件方案和知识。”在各种信息汇聚在一起之后，应从满足需求工程和分析模型（第7～10章）所定义的需求中挑选适合的元素。此时，考虑并取舍候选方案，然后进行聚合，使之成为“构件的某种特定的配置，从而创建最终的产品”[Bel81]。

多样化和聚合需要直觉和判断力，其质量取决于构建类似实体的经验、一系列指导模型演化方式的原则和启发、一系列质量评价的标准以及导出最终设计表示的迭代过程。

新的方法不断出现，分析过程逐步优化，人们对设计的理解也日渐广泛，随之而来的是软件设计的发展和变更^②。即使是今天，大多数软件设计方法都缺少那些更经典的工程设计学科所具有的深度、灵活性和定量性。然而，软件设计的方法是存在的，设计质量的标准是可以获得的，设计表示法也是能够应用的。在本章中，我们将探讨可以应用于所有软件设计的基本概念和原则、设计模型的元素以及模式对设计过程的影响。在第11～14章中，我们将介绍应用于体系结构、接口和构件级设计的多种软件设计方法，也会介绍基于模式和面向Web的设计方法。

11.1 软件工程中的设计

软件设计在软件工程过程中属于核心技术，并且它的应用与所使用的软件过程模型无关。一旦对软件需求进行分析和建模，软件设计就开始了。软件设计是建模活动的最后一个软件工程活动，接着便要进入构建阶段（编码和测试）。

需求模型的每个元素（第8～10章）都提供了创建四种设计模型所必需的信息，这四种设计模型是完整的设计规格说明所必需的。软件设计过程中的信息流如图11-1所示。由基于场景的元素、基于类的元素和行为元素所表示的需求模型是设计任务的输入。使用后续章节所讨论的设计表示法和设计方法，将得到数据或类的设计、体系结构设计、接口设计和构件设计。

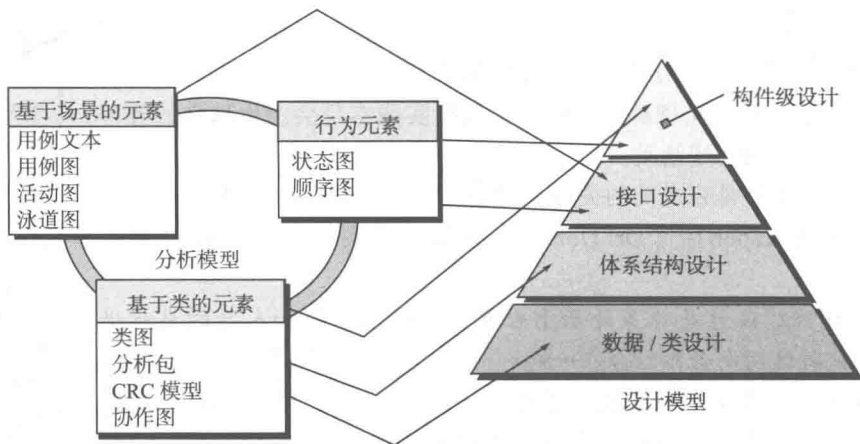


图 11-1 从需求模型到设计模型的转换

② 对软件设计原理感兴趣的读者可能会对 Philippe Kruchten 关于“后现代”设计的有趣讨论感兴趣 [Kru05]。

数据设计或类设计将类模型(第9章)转化为设计类的实现以及软件实现所要求的数据结构。CRC图中定义的对象和关系,以及类属性和其他表示法描述的详细数据内容为数据设计活动提供了基础。在软件体系结构设计中也可能会进行部分类的设计,更详细的类设计则将在设计每个软件构件时进行。

体系结构设计定义了软件的主要结构化元素之间的关系、可满足系统需求的体系结构风格和模式(第12章)以及影响体系结构实现方式的约束[Sha96]。体系结构设计表示可以从需求模型导出,该设计表示基于的是计算机系统的框架。

接口设计描述了软件和协作系统之间、软件和使用人员之间是如何通信的。接口意味着信息流(如数据和控制)和特定的行为类型。因此,使用场景和行为模型为接口设计提供了大量的信息。

构件级设计将软件体系结构的结构化元素变换为对软件构件的过程性描述。从基于类的模型和行为模型中获得的信息是构件设计的基础。

设计过程中所做出的决策将最终影响软件构建的成功与否,更重要的是,会影响软件维护的难易程度。但是,设计为什么如此重要呢?

软件设计的重要性可以用一个词来表达——质量。在软件工程中,设计是质量形成的地方,设计提供了可以用于质量评估的软件表示,设计是将利益相关者的需求准确地转化为最终软件产品或系统的唯一方法。软件设计是所有软件工程活动和随后的软件支持活动的基础。没有设计,将会存在构建不稳定系统的风险,这样的系统稍做改动就无法运行,而且难以测试,直到软件过程的后期才能评估其质量,而那时时间已经不够并且已经花费了大量经费。

建议 软件设计应该总是先考虑数据,数据是所有其他设计元素的基础。基础奠定之后,才能进行体系结构设计。再之后,才能进行其他设计任务。

引述 有两种构建软件设计的方式:一种是使其尽可能简单以致明显没有不足;另一种是使其尽可能复杂以致没有明显的不足。第一种方式更为困难。

C. A. R Hoare

SafeHome 设计与编码

[场景] Jamie 的房间,团队成员准备将需求转化为设计。

[人物] Jamie、Vinod 和 Ed, SafeHome 软件工程团队所有成员。

[对话]

Jamie: 大家都知道, Doug (团队管理者)沉迷于设计。老实说,我真正喜欢的是编码。如果给我 C++ 或者 Java,我会非常高兴。

Ed: 不,你喜欢设计。

Jamie: 你没听我说吗?编码才是我喜欢的。

Vinod: 我想 Ed 的意思是你并不是真的喜欢编码,而是喜欢设计,并喜欢用代码表达设计。代码是你用来表示设计的

语言。

Jamie: 那有什么问题吗?

Vinod: 抽象层。

Jamie: 嗯?

Ed: 程序设计语言有利于表示诸如数据结构和算法的细节,但不利于表示体系结构或者构件之间的协作……就是这个意思。

Vinod: 一个糟糕的体系结构甚至能够摧毁最好的代码。

Jamie(思考片刻): 那么,你们的意思是我不能用代码表示体系结构……这不是事实。

Vinod: 你肯定能在代码中隐含体系结构,但在大部分程序设计语言中,通过检查

代码而快速看到体系结构的全貌是相当困难的。

Ed: 那正是我们在开始编码之前需要的。

Jamie: 我同意, 也许设计和编码不同, 但我仍然更喜欢编码。

11.2 设计过程

软件设计是一个迭代的过程, 通过这个过程, 需求被变换为用于构建软件的“蓝图”。刚开始, 蓝图描述了软件的整体视图, 也就是说, 设计是在高抽象层次上的表达——在该层次上可以直接跟踪到特定的系统目标以及更详细的数据、功能和行为需求。随着设计迭代的开始, 后续的细化导致更低抽象层次的设计表示。这些表示仍然能够跟踪到需求, 但是连接更加错综复杂了。

11.2.1 软件质量指导原则和属性

在整个设计过程中, 我们使用一系列技术评审来评估设计演化的质量。McGlaughlin[McG91] 提出了可以指导良好设计演化的三个特征:

- 设计应当实现所有包含在需求模型中的明确需求, 而且必须满足利益相关者期望的所有隐含需求。
- 对于那些编码者和测试者以及随后的软件维护者而言, 设计应当是可读的、可理解的指南。
- 设计应当提供软件的全貌, 从实现的角度对数据域、功能域和行为域进行说明。

以上每一个特征实际上都是设计过程的目标, 但是如何达到这些目标呢?

质量指导原则。为了评估某个设计表示的质量, 软件团队中的成员必须建立良好设计的技术标准。在 11.3 节中, 我们将讨论设计概念, 这些概念也可以作为软件质量的标准。现在, 考虑下面的指导原则:

1. 设计应展现出这样一种体系结构: (1) 已经使用可识别的体系结构风格或模式创建; (2) 由能够展现出良好设计特征的构件构成 (将在本章后面讨论); (3) 能够以演化的方式^①实现, 从而便于实施与测试。
2. 设计应该模块化, 也就是说, 应将软件逻辑地划分为元素或子系统。
3. 设计应该包含数据、体系结构、接口和构件的清晰表示。
4. 设计应导出数据结构, 这些数据结构适用于要实现的类, 并从可识别的数据模式提取。
5. 设计应导出显示独立功能特征的构件。
6. 设计应导出接口, 这些接口降低了构件之间以及构件与外部环境之间连接的复杂性。
7. 设计的导出应采用可重复的方法进行, 这些方法由软件需求分析过程中获取的信息而产生。
8. 应使用能够有效传达其意义的表示法来表达设计。

① 对于较小的系统, 设计有时也可以线性地进行开发。

引述 编写一段能工作的灵巧的代码是一回事, 而设计能支持某个长久业务的东西则完全是另一回事。

C. Ferguson

提问 良好设计的特征是什么?

引述 设计不仅仅是它看上去以及感觉上如何, 还要看它是如何运作的。

Steve Jobs

满足这些设计原则并非依靠偶然性，而是通过应用基本的设计原则、系统化的方法学和严格的评审来得到保证。

信息栏 评估设计质量——技术评审

设计之所以重要，在于它允许一个软件团队在软件实现前评估软件的质量^①——此时修正错误、遗漏或不一致都不困难且代价不高。但是我们如何在设计过程中评估质量呢？此时，不可能去测试软件，因为还没有可执行的软件，那怎么办？

在设计阶段，可以通过开展一系列的技术评审（Technical Review，TR）来评估质量。技术评审是由软件团队成员召开的会议。通常，根据将要评审的设计信息的范围，选择2~4人参与。每人扮演一个角色：评审组长策划会议、

拟定议程并主持会议；记录员做笔记以保证没有遗漏；制作人是指其工作产品（例如某个软件构件的设计）被评审的人。在会议之前，评审小组的每个成员都会收到一份设计工作产品的拷贝并要求阅读，寻找错误、遗漏或含糊不清的地方。目的是在会议开始时注意到工作产品中的所有问题，以便能够在开始实现该产品之前修正这些问题。技术评审通常持续60~90分钟。评审结束时，评审小组要确认在设计工作产品能够被认可为最终设计模型的一部分之前，是否还需要进一步的行动。

质量属性。Hewlett-Packard[Gra87]制订了一系列的软件质量属性，并取其首字母组合为FURPS，其中各字母分别代表功能性（functionality）、易用性（usability）、可靠性（reliability）、性能（performance）及可支持性（supportability）。FURPS质量属性体现了所有软件设计的目标：

- 功能性通过评估程序的特征集和能力、所提交功能的通用性以及整个系统的安全性来评估。
- 易用性通过考虑人员因素（第6~14章）、整体美感、一致性和文档来评估。
- 可靠性通过测量故障的频率和严重性、输出结果的精确性、平均故障时间（Mean-Time-To-Failure，MTTF）、故障恢复能力和程序的可预见性来评估。
- 性能通过考虑处理速度、响应时间、资源消耗、吞吐量和效率来度量。
- 可支持性综合了可扩展性、可适应性和可用性。这三个属性体现了一个更通用的术语：可维护性。此外，还包括可测试性、兼容性、可配置性（组织和控制软件配置元素的能力，第21章）、系统安装的简易性和问题定位的容易性。

在进行软件设计时，并不是每个软件质量属性都具有相同的权重。有的应用问题可能强调功能性，特别突出安全性；有的应用问题可能要求性能，特别突出处理速度；还有的可能

引述 质量不是那些被束之高阁的观赏品，也不是像圣诞树上的闪亮金箔那样的装饰品。

Robert Pirsig

建议 软件设计师往往注重于问题的解决。不要忘记的是：FURPS属性总是问题的一部分，因此必须要考虑到这些属性。

① 质量因素可以帮助评审小组评估质量。

关注可靠性。抛开权重不谈,重要的是:必须在设计开始时就考虑这些质量属性,而不是在设计完成后和构建已经开始时才考虑。

11.2.2 软件设计的演化

软件设计的演化是一个持续的过程,它已经经历了60多年的发展。早期的设计工作注重模块化程序开发的标准[Den73]和以自顶向下的“结构化”方式对软件结构进行求精的方法([Wir71],[Dah72],[Mil72])。较新的设计方法(如[Jac92],[Gam95])提出了进行设计导出的面向对象方法。近年来,软件体系结构[Kru06]和可用于实施软件体系结构及较低级别设计抽象(如[Hol06],[Sha05])的设计模式已经成为软件设计的重点。面向方面的方法(如[Cla95],[Jac04])、模型驱动开发[Sch06]以及测试驱动开发[Ast04]日益受到重视,这些方法强调在设计中实现更有效的模块化和体系结构的技术。

许多设计方法刚刚被提出,它们从工作中产生,并正被运用于整个行业。正如第8~10章提出的分析方法,每一种软件设计方法引入了独特的启发式和表示法,同时也引入了某种标定软件质量特征的狭隘观点。不过,这些方法都有一些共同的特征:(1)将需求模型转化为设计表示的方法;(2)表示功能性构件及它们之间接口的表示法;(3)细化和分割的启发式方法;(4)质量评估的指导原则。

无论使用哪种设计方法,都应该将一套基本概念运用到数据设计、体系结构设计、接口设计和构件级设计,这些基本概念将在后面几节中介绍。

引述 设计师知道当设计中不再需要减去任何东西,而并不是不再需要增加任何东西的时候,他的设计就很完美了。

Antoine de
St-Expurey

提问 所有设计方法的共同设计特征是什么?

任务集 通用设计任务集

1. 检查信息域模型,并为数据对象及其属性设计合适的数据结构。
2. 使用分析模型选择一种适用于软件的体系结构风格(模式)。
3. 将分析模型分割为若干设计子系统,并在体系结构内分配这些子系统:
 - 确保每个子系统是功能内聚的。
 - 设计子系统接口。
 - 为每个子系统分配分析类或功能。
4. 创建一系列的设计类或构件:
 - 将分析类描述转化为设计类。
 - 根据设计标准检查每个设计类,考虑继承问题。
 - 定义与每个设计类相关的方法和消息。
 - 评估设计类或子系统,并为这些类或子系统选择设计模式。
 - 评审设计类,并在需要时进行修改。
5. 设计外部系统或设备所需要的所有接口。
6. 设计用户接口:
 - 评审任务分析的结果。
 - 基于用户场景对活动序列进行详细说明。
 - 创建接口的行为模型。
 - 定义接口对象和控制机制。
 - 评审接口设计,并根据需要进行修改。
7. 进行构件级设计:
 - 在相对较低的抽象层次上详细描述所有算法。
 - 细化每个构件的接口。
 - 定义构件级的数据结构。
 - 评审每个构件并修正所有已发现的错误。
8. 开发部署模型。

11.3 设计概念

在软件工程的历史上,产生了一系列基本的软件设计概念。尽管多年来人们对于这些概念的关注程度不断变化,但它们都经历了时间的考验。每一种概念都为软件设计者应用更加复杂的设计方法提供了基础。每种方法都有助于:定义一套将软件分割为独立构件的标准,从软件的概念表示中分离出数据结构的细节,为定义软件设计的技术质量建立统一标准。

M. A. Jackson[Jac75]曾经说过:“软件工程师的智慧开始于认识到‘使程序工作’和‘使程序正确’之间的差异。”在后面几节中,将对基本的软件设计概念进行介绍,这些概念为“使程序正确”提供了必要的框架。

11.3.1 抽象

当考虑某一问题的模块化解决方案时,可以给出许多抽象级。在最高的抽象级上,使用问题所处环境的语言以概括性的术语描述解决方案。在较低的抽象级上,将提供更详细的解决方案说明。当力图陈述一种解决方案时,面向问题的术语和面向实现的术语会同时使用。最后,在最低的抽象级上,以一种能直接实现的方式陈述解决方案。

在开发不同层次的抽象时,软件设计师力图创建过程抽象和数据抽象。过程抽象是指具有明确和有限功能的指令序列。“过程抽象”这一命名暗示了这些功能,但隐藏了具体的细节。过程抽象的例子如开门。开隐含了一长串的过程性步骤(例如,走到门前,伸出手并抓住把手,转动把手并拉门,从移动门走开等)。^①

数据抽象是描述数据对象的具名数据集合。在过程抽象开的场景下,我们可以定义一个名为 door 的数据抽象。同任何数据对象一样,door 的数据抽象将包含一组描述门的属性(例如,门的类型、转动方向、开门方法、重量和尺寸)。因此,过程抽象开将利用数据抽象 door 的属性中所包含的信息。

11.3.2 体系结构

软件体系结构意指“软件的整体结构和这种结构为系统提供概念完整性的方式”[Sha95a]。从最简单的形式来看,体系结构是程序构件(模块)的结构或组织、这些构件交互的方式以及这些构件所用数据的结构。然而在更广泛的意义上,构件可以概括为主要的系统元素及其交互方式的表示。

软件设计的目标之一是导出系统体系结构示意图,该示意图作为一个框架,将指导更详细的设计活动。一系列的体系结构模式使软件工程师能够重用设计层概念。

Shaw 和 Garlan[Sha95a]描述了一组属性,这组属性应该作为体系结构设计的一部分进行描述。结构特性定义了“系统的构件(如模块、对象、过滤器)、构件被封装的方式以及构件之间相互作用的方式”。外部功能特

引述 抽象是人类处理复杂问题的基本方法之一。

Grady Booch

建议 作为设计师,致力于得到解决现有问题的过程抽象和数据抽象,但如果他们能在整个问题域中起作用,那就更好了。

网络资源 关于软件体系结构深入的讨论可以在 www.sei.cmu.edu/ata/ata_init.html 找到。

引述 软件体系结构是在质量、进度和成本方面具有最高投资回报的工作产品。

Len Bass et al.

^① 然而,需要注意的是:只要过程抽象隐含的功能相同,一组操作就可以被另一组操作代替。因此,如果门是自动的并连接到传感器上,那么实现“开”所需的步骤将会完全不同。

性指出“设计体系结构如何满足需求，这些需求包括性能需求、能力需求、可靠性需求、安全性需求、可适应性需求以及其他系统特征需求”。相关系统族“抽取出相似系统设计中常遇到的重复性模式”。

一旦给出了这些特性的规格说明，就可以用一种或多种不同的模型来表示体系结构设计[Gar95]。结构模型将体系结构表示为程序构件的有组织的集合。框架模型可以通过确定相似应用中遇到的可复用体系结构设计框架（模式）来提高设计抽象的级别。动态模型强调程序体系结构的行为方面，指明结构或系统配置如何随着外部事件的变化而产生变化。过程模型强调系统必须提供的业务或技术流程的设计。最后，功能模型可用于表示系统的功能层次结构。

为了表示以上描述的模型，人们已经开发了许多不同的体系结构描述语言（Architectural Description Language, ADL）[Sha95b]。尽管提出了许多不同的ADL，但大多数ADL都提供描述系统构件和构件之间相互联系方式的机制。

需要注意的是，关于体系结构在设计中的地位还存在一些争议。一些研究者主张将软件体系结构的设计从设计中分离出来，并在需求工程活动和更传统的设计活动之间进行。另外一些研究者认为体系结构设计是设计过程不可分割的一部分。第12章将讨论软件体系结构特征描述方式及软件体系结构在设计中的作用。

11.3.3 模式

Brad Appleton 以如下方式定义设计模式：“模式是具名的洞察力财宝，对于竞争事件中某确定环境下重复出现的问题，它承载了已证实的解决方案的精髓”[App00]。换句话说，设计模式描述了解决某个特定设计问题的设计结构，该设计问题处在一个特定环境中，该环境会影响到模式的应用和使用方式。

每种设计模式的目的是提供一种描述，以使设计人员可以决定：（1）模式是否适用于当前的工作；（2）模式是否能够复用（因此节约设计时间）；（3）模式是否能够用于指导开发一个相似的但功能或结构不同的模式。

11.3.4 关注点分离

关注点分离是一个设计概念[Dij82]，它表明任何复杂问题如果被分解为可以独立解决或优化的若干块，该复杂问题便能够更容易地得到处理。关注点是一个特征或一个行为，被指定为软件需求模型的一部分。将关注点分割为更小的关注点（由此产生更多可管理的块），便可用更少的工作量和时间解决一个问题。

另一个结果是：两个问题被结合到一起的认知复杂度经常高于每个问题各自的认知复杂度之和。这就引出了“分而治之”的策略——把一个复杂问题分解为若干可管理的块来求解时将会更容易。这对于软件的模块化具有重要的意义。

关注点分离在其他相关设计概念中也有体现：模块化、方面、功能独立、求精。每个概念都会后面的小节中讨论。

11.3.5 模块化

模块化是关注点分离最常见的表现。软件被划分为独立命名的、可处理的构件，有时被

引述 每个模式都描述了一个在我们所处环境内反复发生的问题，然后描述该问题的核心解决方案，你可以几百万次地重复使用该解决方案，而根本不需要用同样的方式重复工作两次。

Christopher
Alexander

称为模块，把这些构件集成到一起可以满足问题的需求。

有人提出“模块化是软件的单一属性，它使程序能被智能化地管理”[Mye78]。软件工程师难以掌握单块软件（即由一个单独模块构成的大程序）。对于单块大型程序，其控制路径的数量、引用的跨度、变量的数量和整体的复杂度使得理解这样的软件几乎是不可能的。绝大多数情况下，为了理解更容易，都应当将设计划分成许多模块，这样做的结果是降低构建软件所需的成本。

回顾关于关注点分离的讨论，可以得出结论：如果无限制地划分软件，那么开发软件所需的工作量将会小到忽略不计！不幸的是，其他因素开始起作用，导致这个结论是不成立的（十分遗憾）。如图 11-2 所示，开发单个软件模块的工作量（成本）的确随着模块数的增加而下降。给定同样的需求，更多的模块意味着每个模块的规模更小。然而，随着模块数量的增加，集成模块的工作量（成本）也在增加。这些特性形成了图示中的总体成本或工作量曲线。事实上，的确存在一个模块数量 M ，这个数量可以带来最小的开发成本。但是，我们缺乏成熟的技术来精确地预测 M 。

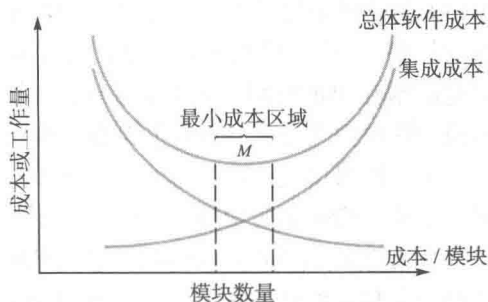


图 11-2 模块化和软件成本

在考虑模块化的时候，图 11-2 所示的曲线确实提供了有益的指导。在进行模块化的时候，应注意保持在 M 附近，避免不足的模块化或过度的模块化问题。但是如何知道 M 的附近在哪里呢？怎样将软件划分成模块呢？回答这些问题需要理解本章后面提出的其他设计概念。

提问 对于一个给定的系统，合适的模块数量是多少？

模块化设计（以及由其产生的程序）使开发工作更易于规划；可以定义和交付软件增量；更容易实施变更；能够更有效地开展测试和调试；可以进行长期维护而没有严重的副作用。

11.3.6 信息隐蔽

模块化概念面临的一个基本问题是：“应当如何分解一个软件解决方案以获得最好的模块集合？”信息隐蔽原则 [Par72] 建议模块应该“具有的特征是：每个模块对其他所有模块都隐蔽自己的设计决策”。换句话说，模块应该被特别说明并设计，使信息（算法和数据）都包含在模块内，其他模块无需对这些信息进行访问。

隐蔽的含义是，通过定义一系列独立的模块得到有效的模块化，独立模块之间只交流实现软件功能所必需的信息。抽象有助于定义构成软件的过程（或信息）实体。隐蔽定义并加强了对模块内过程细节的访问约束以及对模块所使用的任何局部数据结构的访问约束 [Ros75]。

将信息隐蔽作为系统模块化的一个设计标准，在测试过程以及随后的软件维护过程中需要进行修改时，将使我们受益匪浅。由于大多数数据和过程对软件的其他部分是隐蔽的，因此，在修改过程中不小心引入的错误就不太可能传播到软件的其他地方。

关键点 信息隐蔽的目的是将数据结构和处理过程的细节隐藏在模块接口之后。用户不需要了解模块内部的具体细节。

11.3.7 功能独立

功能独立的概念是关注点分离、模块化、抽象和信息隐蔽概念的直接产物。在关于软件

设计的一篇里程碑性的文章中, Wirth[Wir71] 和 Parnas[Par72] 间接提到增强模块独立性的细化技术。Stevens、Myers 和 Constantine[Ste74] 等人在其后又巩固了这一概念。

通过开发具有“专一”功能和“避免”与其他模块过多交互的模块, 可以实现功能独立。换句话说, 软件设计时应使每个模块仅涉及需求的某个特定子集, 并且当从程序结构的其他部分观察时, 每个模块只有一个简单的接口。

人们会提出一个很合理的问题: 独立性为什么如此重要? 具有有效模块化(也就是独立模块)的软件更容易开发, 这是因为功能被分隔而且接口被简化(考虑由一个团队进行开发时的结果)。独立模块更容易维护(和测试), 因为修改设计或修改代码所引起的副作用被限制, 减少了错误扩散, 而且模块复用也成为可能。概括地说, 功能独立是良好设计的关键, 而设计又是软件质量的关键。

独立性可以通过两条定性的标准进行评估: 内聚性和耦合性。内聚性显示了某个模块相关功能的强度; 耦合性显示了模块间的相互依赖性。

内聚性是 11.3.6 节说明的信息隐蔽概念的自然扩展。一个内聚的模块执行一个独立的任务, 与程序的其他部分构件只需要很少的交互。简单地说, 一个内聚的模块应该只完成一件事情(理想情况下)。即使我们总是争取高内聚性(即专一性), 一个软件构件执行多项功能也经常是必要的和可取的。然而, 为了实现良好的设计, 应该避免“分裂型”构件(执行多个无关功能的构件)。

耦合性表明软件结构中多个模块之间的相互连接。耦合性依赖于模块之间的接口复杂性、引用或进入模块所在的点以及什么数据通过接口进行传递。在软件设计中, 应当尽力得到最低可能的耦合。模块间简单的连接性使得软件易于理解并减少“涟漪效果”(ripple effect)的倾向[Ste74]。当在某个地方发生错误并传播到整个系统时, 就会引起“涟漪效果”。

11.3.8 求精

逐步求精是一种自顶向下的设计策略, 最初由 Niklaus Wirth[Wir71] 提出。通过连续细化过程细节层进行应用开发, 通过逐步分解功能的宏观陈述(过程抽象)进行层次开发, 直至最终到达程序设计语言的语句这一级。

求精实际上是一个细化的过程。该过程从高抽象级上定义的功能陈述(或信息描述)开始。也就是说, 该陈述概念性地描述了功能或信息, 但是没有提供有关功能内部的工作或信息内部的结构。可以在原始陈述上进行细化, 随着每次细化的持续进行, 将提供越来越多的细节。

抽象和细化是互补的概念。抽象能够明确说明内部过程和数据, 但对“外部使用者”隐藏了低层细节; 细化有助于在设计过程中揭示低层细节。这两个概念均有助于设计人员在设计演化中构建出完整的设计模型。

11.3.9 方面

当我们开始进行需求分析时, 一组“关注点”就出现了。这些关注点“包括需求、用例、特征、数据结构、服务质量问题、变量、知识产权边界、协作、模式以及合同”[AOS07]。理想情况下, 可以按某种方式组织需求模型, 该方式允许分离每个关注点(需求), 使得我

提问 为什么我们总是努力构造独立模块?

关键点 内聚性是一个模块对于一件事情侧重程度的定性指标。

关键点 耦合性是一个模块和其他模块及外部世界连接程度的定性指标。

建议 有一种趋势是直接进行全面详细的设计, 而跳过细化步骤, 这将导致错误和遗漏, 并使得设计更难于评审。因此, 一定要实施逐步求精。

们能够独立考虑每个关注点(需求)。然而实际上,某些关注点跨越了整个系统,从而很难进行分割。

开始进行设计时,需求被细化为模块设计表示。考虑两个需求,A和B。“如果已经选择了一种软件分解(细化),在这种分解中,如果不考虑需求A的话,需求B就不能得到满足”[Ros04],那么需求A横切需求B。

例如,考虑www.safehomeassured.com网站应用中的两个需求。用第8章中讨论的用例ACS-DCV描述需求A,设计求精将集中于那些能够使注册用户通过放置在空间中的摄像机访问视频的模块。需求B是一个通用的安全需求,要求注册用户在使用www.safehomeassured.com之前必须先进行验证,该需求用于SafeHome注册用户可使用的所有功能中。当设计求精开始的时候,A*是需求A的一个设计表示,B*是需求B的一个设计表示。因此,A*和B*是关注点的表示,且B*横切A*。

方面是一个横切关注点的表示,因此,需求注册用户在使用www.safehomeassured.com之前必须先进行验证的设计表示B*是SafeHome网站应用的一个方面。标识方面很重要,以便于在开始求精和模块化的时候,设计能够很好地适应这些方面。在理想情况下,一个方面作为一个独立的模块(构件)进行实施,而不是作为“分散的”或者和许多构件“纠缠的”软件片断进行实施[Ban06]。为了做到这一点,设计体系结构应当支持定义一个方面,该方面即一个模块,该模块能够使该关注点经过它横切的所有其他关注点而得到实施。

11.3.10 重构

很多敏捷方法(第5章)都建议一种重要的设计活动——重构,重构是一种重新组织的技术,可以简化构件的设计(或代码)而无需改变其功能或行为。Fowler[Fow00]这样定义重构:“重构是使用这样一种方式改变软件系统的过程:不改变代码(设计)的外部行为而是改进其内部结构。”

在重构软件时,检查现有设计的冗余性、没有使用的设计元素、低效的或不必要的算法、拙劣的或不恰当的数据结构以及其他设计不足,修改这些不足以获得更好的设计。例如,第一次设计迭代可能得到一个构件,表现出很低的内聚性(即执行三个功能但是相互之间仅有有限的联系)。在深思熟虑之后,设计人员可能决定将原构件重新分解为三个独立的构件,其中每个构件都表现出更高的内聚性。结果则是,软件更易于集成、测试与维护。

尽管重构的目的是以某种方式修改代码,而并不改变它的外部行为,但意外的副作用可能发生,并且也确实会发生。为此,可以使用重构工具[Soa10]自动分析变更,并“生成可用于检测行为变更的测试套件”。

11.3.11 面向对象的设计概念

面向对象(Object-Oriented, OO)范型广泛应用于现代软件工程。附录2是为那些不熟悉面向对象概念(如类、对象、继承、消息和多态等)的读者提供的。

引述 时常浏览一下封面,确保它不是一本关于软件设计的书,这样你会更容易读懂书中的“魔法”原理。

Bruce Tognazzini

关键点 横切关注点是系统的某个特征,它适用于许多不同的需求。

网络资源 重构的优秀资源可以在网站www.refactoring.com找到。

网络资源 大量重构模式可以在网站<http://c2.com/cgi/wiki?RefactoringPatterns>找到。

SafeHome 设计概念

[场景] Vinod 的房间, 设计建模开始。

[人物] Vinod、Jamie 和 Ed, SafeHome 软件工程团队成员; 还有 Shakira, 团队的新成员。

[对话]

(这四个团队成员上午都参加了一位本地计算机科学教授举行的名为“应用基本的设计概念”的研讨会, 他们刚从会上回来。)

Vinod: 你们从研讨会学到什么没有?

Ed: 大部分的东西我都已经知道, 但我想重温一遍总不是什么坏事。

Jamie: 我在计算机专业学习时, 从没有真正理解信息隐蔽为什么像他们说得那么重要。

Vinod: 因为……底线……这是减少错误在程序内扩散的一种技术。实际上, 功能独立做的也是同样的事。

Shakira: 我不是计算机科学专业的, 因此教授提到的很多东西对我而言都是新的。我能生成好的代码而且速度快, 我不明白这个东西为什么这么重要。

Jamie: 我了解你的工作, Shak, 你要知道, 其实你是在自然地做这些事情……这就是

为什么你的设计和编码很有效。

Shakira (微笑): 是的, 我通常的确是尽量将代码分割, 让分割后的代码关注于一件事, 保持接口简单而且有约束, 在任何可能的时候重用代码……就是这样。

Ed: 模块化、功能独立、隐蔽、模式……现在明白了。

Jamie: 我至今还记得我上的第一节编程课……他们教我们用迭代方式细化代码。

Vinod: 设计可以采用同样的方式, 你知道的。

Jamie: 我以前从未听说过的概念是“方面”和“重构”。

Shakira: 我记得她说那是用在极限编程中的。

Ed: 是的。其实它和细化并没有太大不同, 它只是在设计或代码完成后进行。我认为, 这是软件开发过程中的一种优化。

Jamie: 让我们回到 SafeHome 设计。我觉得在我们开发 SafeHome 的设计模型时, 应该将这些概念用在评审检查单上。

Vinod: 我同意。但重要的是, 我们都要能够在设计时想一想这些概念。

11.3.12 设计类

分析模型定义了一组分析类(第9章), 每一个分析类都描述问题域中的某些元素, 这些元素关注用户可见的问题方面。分析类的抽象级相对较高。

当设计模型发生演化时, 必须定义一组设计类, 它们可以: (1) 通过提供设计细节对分析类进行求精, 而这些设计细节将促成类的实现; (2) 实现支持业务解决方案的软件基础设施。以下给出了五种不同类型的设计类[Amb01], 每一种都表示设计体系结构的一个不同层次: (1) 用户接口类, 定义人机交互(Human-Computer Interaction, HCI)所必需的所有抽象, 并且经常在隐喻的环境中实施HCI; (2) 业务域类, 识别实现某些业务域元素所必需的属性和服务(方法), 通过一个或更多的分析类进行定义; (3) 过程类, 实现完整的管理业务域类所必需的低层业务抽象; (4) 持久类, 用于表示将在软件执行之外持续存在的数据存储(例如, 数据库); (5) 系统类, 实现软件管理和控制功能, 使得

提问 设计者要创建哪些类型的类?

系统能够运行，并在其计算环境内与外界通信。

随着体系结构的形成，每个分析类（第9章）转化为设计表示，抽象级就降低了。也就是说，分析类使用业务域的专门用语描述数据对象（以及数据对象所用的相关服务）。设计类更多地表现技术细节，用于指导实现。

Arlow 和 Neustadt[Arl02] 给出建议：应当对每个设计类进行评审，以确保设计类是“组织良好的”(well-formed)。他们为组织良好的设计类定义了四个特征。

完整性与充分性。设计类应该完整地封装所有可以合理预见的（根据对类名的理解）存在于类中的属性和方法。例如，为视频编辑软件定义的 Scene 类，只有包含与创建视频场景相关的所有合理的属性和方法时，它才是完整的。充分性确保设计类只包含那些“对实现该类的目的是足够”的方法，不多也不少。

提问 什么才是“组织良好的”设计类？

原始性。和一个设计类相关的方法应该关注于实现类的某一个服务。一旦服务已经被某个方法实现，类就不应该再提供完成同一事情的另外一种方法。例如，视频编辑软件的 VideoClip 类，可能用属性 start-point 和 end-point 指定剪辑的起点和终点（注意，加载到系统的原始视频可能比要用的部分长）。方法 setStartPoint() 和 setEndPoint() 为剪辑提供了设置起点和终点的唯一手段。

高内聚性。一个内聚的设计类具有小的、集中的职责集合，并且专注于使用属性和方法来实现那些职责。例如，视频编辑软件的 VideoClip 类可能包含一组用于编辑视频剪辑的方法。只要每个方法只关注于和视频剪辑相关的属性，内聚性就得以维持。

低耦合性。在设计模型内，设计类之间相互协作是必然的。但是，协作应该保持在一个可以接受的最小范围内。如果设计模型高度耦合（每一个设计类都和其他所有的设计类有协作关系），那么系统就难以实现、测试，并且维护也很费力。通常，一个子系统内的设计类对其他子系统内的类应仅有有限的了解。该限制被称作 Demeter 定律 [Lie03]，该定律建议一个方法应该只向周边类中的方法发送消息。[⊖]

SafeHome 将分析类细化为设计类

[场景] Ed 的房间，开始进行设计建模。

[人物] Vinod 和 Ed, SafeHome 软件工程团队成员。

[对话]

(Ed 正进行 FloorPlan 类的设计工作（参考 9.3 节的讨论以及图 9-2），并进行设计模型的细化。)

Ed: 你还记得 FloorPlan 类吗？这个类用作监视和住宅管理功能的一部分。

Vinod (点头): 是的，我好像想起来我们把它用作住宅管理 CRC 讨论的一部分。

Ed: 确实如此。不管怎样，我们要对设计进行细化，希望显示出我们将如何真正地实现 FloorPlan 类。我的想法是把它实现为一组链表（一种特定的数据结构）。像这样……我必须细化分析类 FloorPlan（图 9-2），实际上，是它的一种简化。

Vinod: 分析类只显示问题域中的东西，也就是说，在电脑屏幕上实际显示的、最终用户可见的那些东西，对吗？

Ed: 是的。但对于 FloorPlan 设计类来说，我已经开始添加一些实现中特有的东西。

[⊖] Demeter 定律的一种非正式表述是：“每个单元应该只和它的朋友谈话，不要和陌生人谈话。”

需要说明的是 FloorPlan 是段 (Segment 类) 的聚集, Segment 类由墙段、窗户、门等的列表组成。Camera 类和 FloorPlan 类协作, 这很显然, 因为在平面图中可以有任意多摄像机。

Vinod: 咳, 让我们看看新的 FloorPlan 设计类图。

(Ed 向 Vinod 展示图 11-3。)

Vinod: 好的, 我看得出来你想做什么了。这样你能够很容易地修改平面图, 因为新的东西可以在列表 (聚集) 中添加或删除, 而不会有任何问题。

Ed (点头): 是的, 我认为这样是可以的。

Vinod: 我也赞同。

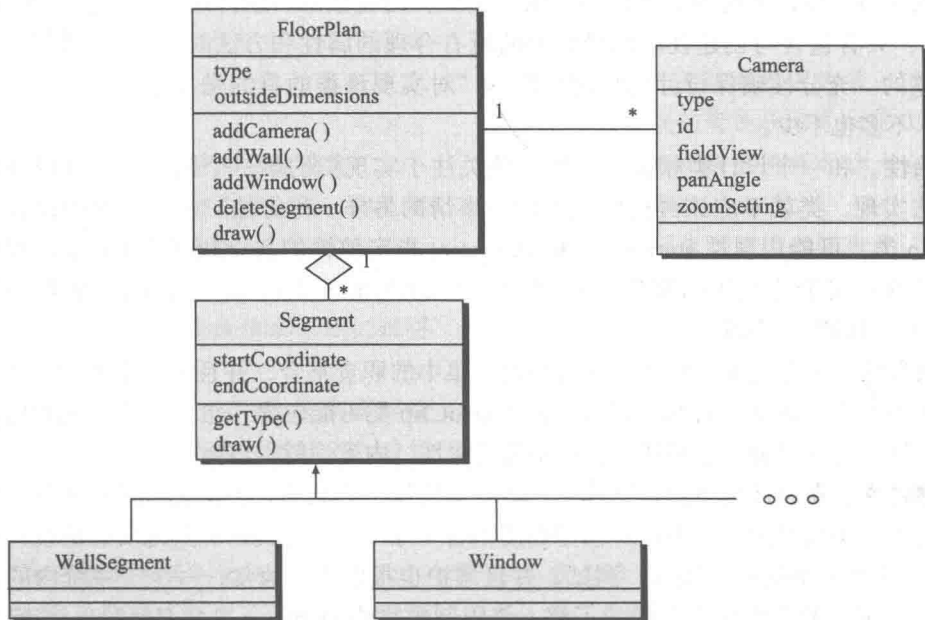


图 11-3 FloorPlan 的设计类和类的复合聚集

11.3.13 依赖倒置

许多旧一些的软件体系结构是层次结构。在体系结构的顶层, “控制” 构件依赖于较低层的 “工作者” 构件, 以完成各种内聚任务。比如, 考虑一个具有三个组件的简单程序, 该程序的目的是读取键盘的按键并将结果输出到打印机。控制模块 C 需要协调另外两个模块——按键读取模块 R 和写入打印机模块 W。

由于控制模块 C 高度依赖于 R 和 W, 因此程序设计是耦合的。为了消除这种依赖, “工作者” 模块 R 和 W 应当通过抽象由控制模块 C 调用。在面向对象的软件工程中, 抽象被作为抽象类 R* 和 W* 以获得实现。然后, 这些抽象类可以用来调用执行读写功能的工作者类。因此, 一个 copy 类 C 调用抽象类 R* 和 W*, 抽象类指向合适的工作者类 (例如, R* 类可能指向一个环境中 keyboard 类的 read() 操作和另一个环境中 sensor 类的 read() 操作)。这种方法降低了耦合性, 并提高了设计的可测性。

提问 依赖倒置的原则是什么?

前段中讨论的例子可以与依赖倒置原则一概而论 [Obj10], 该原则这样描述: 高层模块

(类) 不应当(直接)依赖于低层模块, 两者都应当依赖于抽象。抽象不应当依赖于细节, 细节应当依赖于抽象。

11.3.14 测试设计

到底应当先开始软件设计还是测试用例设计, 这是一个争论不休的鸡与蛋的问题。Rebecca Wirfs-Brock[Wir09]写道:

测试驱动开发的倡导者们在编写任何其他代码之前先编写测试代码。他们谨记 Peter 的信条——测试要快, 失败要快, 调整要快。他们以一种简短而快速的方式实施周期演变, 编写测试代码——测试未通过——编写足够的代码以通过测试——测试通过, 在这一过程中, 测试指导他们的设计。

但如果先开始进行设计, 那么设计(以及编码)必须带有一些接缝。所谓接缝, 即详细设计中的一些位置, 在这些位置可以“插入一些测试代码, 这些代码可用以探测运行中软件的状态”, 以及“将待测试的代码从它的产生环境中分离出来, 以便在受控的测试环境中执行这些代码”[Wir09]。

有时候, 接缝也被称为“测试沟”, 它们必须被有意识地在构件级进行设计。为了实现这一点, 设计者必须考虑将用于演练构件的测试。正如 Wirf-Brock 所述: “简言之, 需要提供适当的测试启示——以一种方式将设计分解为若干因素, 测试代码可以询问并控制运行中的系统。”

引述 测试要快, 失败要快, 调整要快。

Tom Peters

11.4 设计模型

可以从两个不同的维度观察设计模型, 如图 11-4 所示。过程维度表示设计模型的演化, 设计任务作为软件过程的一部分被执行。抽象维度表示详细级别, 分析模型的每个元素转化为一个等价的设计, 然后迭代求精。参考图 11-4, 虚线表示分析模型和设计模型之间的边界。在某些情况下, 分析模型和设计模型之间可能存在明显的差异; 而有些情况下, 分析模型慢慢地融入设计模型而没有明显的差异。

设计模型的元素使用了很多 UML 图^①, 有些 UML 图在分析模型中也会用到。差别在于这些图被求精和细化为设计的一部分, 并且提供了更具体的实施细节, 突出了体系结构的结构和风格、体系结构中的构件、构件之间以及构件和外界之间的接口。

然而, 要注意到, 沿横轴表示的模型元素并不总是顺序开发的。大多数情况下, 初步的体系结构设计是基础, 随后是接口设计和构件级设计(通常是并行进行)。通常, 直到设计全部完成后才开始部署模型的工作。

在设计过程中的任何地方都可以应用设计模式。这些模式能够使设计人员将设计知识应用到他人已经遇到并解决了的特定领域问题中。

关键点 设计模型有 4 个主要元素: 数据、体系结构、构件和接口。

引述 提出“设计是否是必要的”或“能否负担得起”这样的问题非常离题, 因为设计是不可避免的。不是好的设计就是坏的设计, 根本不可能不要设计。

Douglas Martin

① 附录 1 提供了基本 UML 概念和符号的使用手册。

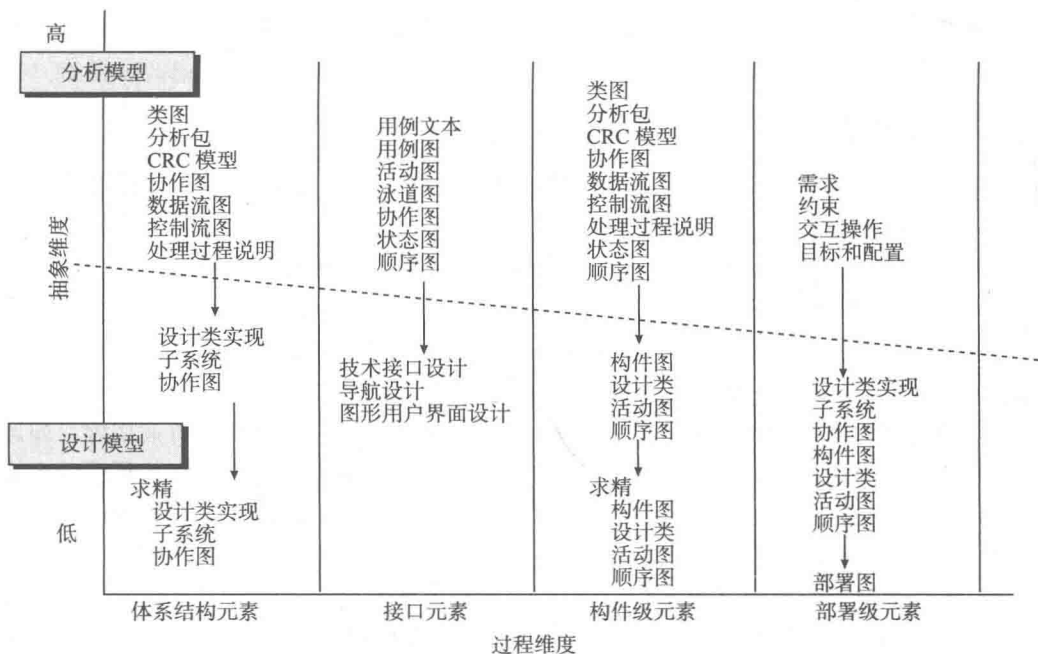


图 11-4 设计模型的维度

11.4.1 数据设计元素

和其他软件工程活动一样，数据设计（有时也称为数据体系结构）创建在了高抽象级上（以客户或用户的数据观点）表示的数据模型和信息模型。之后，数据模型被逐步求精为特定实现的表示，亦即计算机系统能够处理的表示。在很多软件应用中，数据体系结构对于必须处理该数据的软件的体系结构将产生深远的影响。

数据结构通常是软件设计的重要部分。在程序构件级，数据结构设计以及处理这些数据的相关算法对于创建高质量的应用程序是至关重要的。在应用级，从数据模型（源自于需求工程）到数据库的转变是实现系统业务目标的关键。在业务级，收集存储在不同数据库中的信息并重新组织为“数据仓库”要使用数据挖掘或知识发现技术，这些技术将会影响到业务本身的成功。在每一种情况下，数据设计都发挥了重要作用。第 12 章将更详细地讨论数据设计。

关键点 在体系结构（应用）级，数据设计关注文件或数据库；在构件级，数据设计考虑实现局部数据对象所需的数据结构。

11.4.2 体系结构设计元素

软件的体系结构设计等效于房屋的平面图。平面图描绘了房间的整体布局，包括各房间的尺寸、形状、相互之间的联系，能够进出房间的门窗。平面图为我们提供了房屋的整体视图；而体系结构设计元素为我们提供了软件的整体视图。

体系结构模型 [Sha96] 从以下三个来源导出：（1）关于将要构建的软件的应用域信息；（2）特定的需求模型元素，如数据流图或分析类、现有问题中它们的关系和协作；（3）可获得的体系结构风格（第 12 章）和模式。

体系结构设计元素通常被描述为一组相互联系的子系统，且常常从需

引述 你可以在绘图桌上使用橡皮或在建筑工地现场使用大铁锤。

Frank Lloyd Wright

求模型中的分析包中派生出来。每个子系统有其自己的体系结构（如图形用户界面可能根据之前存在的用户接口体系结构进行了结构化）。体系结构模型特定元素的导出技术将在第12章中介绍。

11.4.3 接口设计元素

软件的接口设计相当于一组房屋的门、窗和外部设施的详细绘图（以及规格说明）。门、窗、外部设施的详细图纸（以及规格说明）作为平面图的一部分，大体上告诉我们：事件和信息如何流入和流出住宅以及如何在平面图的房间内流动。软件接口设计元素描述了信息如何流入和流出系统，以及被定义为体系结构一部分的构件之间是如何通信的。

接口设计有三个重要的元素：（1）用户界面（User Interface, UI）；（2）和其他系统、设备、网络、信息生成者或使用者的外部接口；（3）各种设计构件之间的内部接口。这些接口设计元素能够使软件进行外部通信，还能使软件体系结构中的构件之间进行内部通信和协作。

UI设计（越来越多地被称作可用性设计）是软件工程中的主要活动，这会在第14章中详细地考虑。可用性设计包含美学元素（例如，布局、颜色、图形、交互机制）、人机工程元素（例如，信息布局、隐喻、UI导航）和技术元素（例如，UI模式、可复用构件）。通常，UI是整个应用体系结构内独一无二的子系统。

外部接口设计需要发送和接收信息实体的确定信息。在所有情况下，这些信息都要在需求工程（第7章）过程中进行收集，并且在接口^①设计开始时进行校验。外部接口设计应包括错误检查和适当的安全特征检查。

内部接口设计和构件级设计（第13章）紧密相关。分析类的设计实现呈现了所有需要的操作和消息传递模式，使得不同类的操作之间能够进行通信和协作。每个消息的设计必须提供必不可少的信息传递以及所请求操作的特定功能需求。

在有些情况下，接口建模的方式和类所用的方式几乎一样。在UML中，接口如下定义[OMG03a]：“接口是类、构件或其他分类符（包括子系统）的外部可见的（公共的）操作说明，而没有内部结构的规格说明。”更简单地说，接口是一组描述类的部分行为的操作，并提供了这些操作的访问方法。

例如，SafeHome安全功能使用控制面板，控制面板允许户主控制安全功能的某些方面。在系统的高级版本中，控制面板的功能可能会通过移动平台（例如智能手机或平板电脑）实现。

ControlPanel类（图11-5）提供了和键盘相关的行为，因此必须实现操作readKeyStroke()和decodeKey()。如果这些操作提供给其他类（在此例中是Tablet和SmartPhone），定义如图11-5所示的接口是非常有用的。名为KeyPad的接口表示为<<interface>>构造型（stereotype），或用一个

引述 与良好的设计相比，公众更熟悉拙劣的设计。实际上，公众更习惯拙劣的设计，因为生活就是如此。新的有危险，而旧的更让人安心。

Paul Rand

关键点 接口设计元素有三部分：用户接口、系统和外部应用的接口、应用系统内部构件之间的接口。

引述 现在的每一样东西以后都会消失。稍微放松一下，再返回到你的工作中，你的判断将更可靠。因为离开一定距离，工作看起来更小，更容易远瞰，更容易发现和谐和比例的缺失。

Leonardo DaVinci

网络资源 有关UI设计非常有用的信息可以在www.useit.com找到。

① 接口特征可能随时间变化。因此，设计者应当确保接口的规格说明是准确且完整的。

带有标识且用一条线和类相连的小圆圈表示，定义接口时并没有实现键盘行为所必需的属性和操作集合。

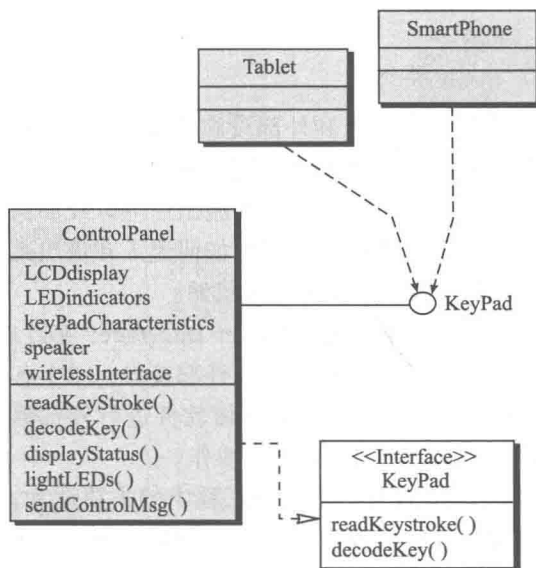


图 11-5 ControlPanel 的接口表示

带有三角箭头的虚线（图 11-5）表示 ControlPanel 类提供了 KeyPad 操作以作为其行为的一部分。在 UML 中，这被称为实现。也就是说，ControlPanel 行为的一部分将通过实现 KeyPad 操作来实现。这些操作将被提供给那些访问该接口的其他类。

引述 在设计傻瓜级的东西时，人们常犯的一个错误是低估傻瓜的聪明。

Douglas Adams

11.4.4 构件级设计元素

软件的构件级设计相当于一个房屋中每个房间的一组详图（以及规格说明）。这些图描绘了每个房间内的布线和管道、电器插座和墙上开关、水龙头、水池、淋浴、浴盆、下水管道、壁橱和储藏室的位置，以及房间相关的任何其他细节。

软件的构件级设计完整地描述了每个软件构件的内部细节。为此，构件级设计为所有局部数据对象定义数据结构，为所有在构件内发生的处理定义算法细节，并定义允许访问所有构件操作（行为）的接口。

在面向对象的软件工程中，使用 UML 图表现的一个构件如图 11-6 所示。图中表示的构件名为 SensorManagement（SafeHome 安全功能的一部分）。虚线箭头连接了构件和名为 Sensor 的类。SensorManagement 构件完成所有和 SafeHome 传感器相关的功能，包括监控和配置传感器。第 13 章将进一步讨论构件图。

引述 细节并不仅仅是细节，细节构成设计。

Charles Eames

构件的设计细节可以在很多不同的抽象级进行建模。UML 活动图可用来表示处理逻辑，构件详细的过程流可以使用伪代码表示（类似编程语言的表示方法，在第 13 章讨论），也可以使用一些图形（例如流程图或盒图）来表示。算法结构

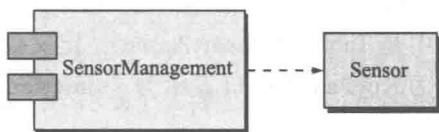


图 11-6 UML 构件图

遵守结构化编程的规则（即一套约束程序构造）。基于待处理数据对象的特性所选择的数据结构，通常会使用伪代码或程序语言进行建模，以便将之用于实施。

11.4.5 部署级设计元素

部署级设计元素指明软件功能和子系统将如何在支持软件的物理计算环境内进行分布。例如，SafeHome 产品元素被配置在三种主要的计算环境内运行——基于住宅的 PC、SafeHome 控制面板和位于 CPI 公司的服务器（提供基于 Internet 的系统访问）。此外，移动平台也可以提供有限的功能。

在设计过程中，开发的 UML 部署图以及随后的细化如图 11-7 所示。图中显示了 3 种计算环境（实际上，还可能包括传感器、摄像机和移动平台传送的功能）。图中标识出了每个计算元素中还有子系统（功能）。例如，个人计算机中有完成安全、监视、住宅管理和通信功能的子系统。此外，还设计了一个外部访问子系统，以管理外界资源对 SafeHome 系统的访问。每个子系统需要进行细化，用以说明该子系统所实现的构件。

如图 11-7 所示，图中使用了描述符形式，这意味着部署图表明了计算环境，但并没有明确地说明配置细节。例如，“个人计算机”并没有进一步地明确它是一台 Mac PC、一台基于 Windows 的 PC、Linux 系统还是带有相关操作系统的移动平台。在设计后续阶段或构建开始时，需要用实例形式重新为部署图提供这些细节，明确每个实例的部署（专用的称为硬件配置）。

关键点 部署图刚开始使用描述符形式，粗略描述部署环境。后来使用实例形式，明确描述配置的元素。

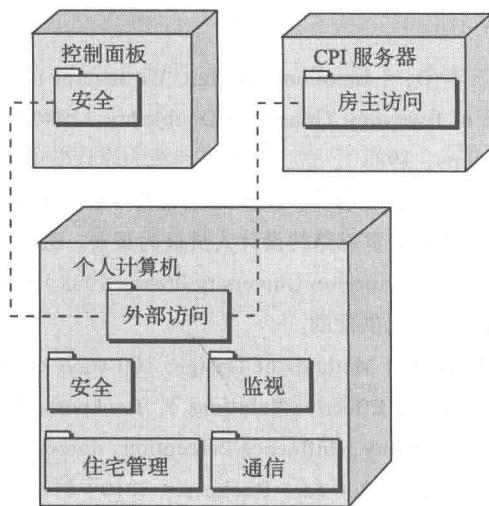


图 11-7 UML 部署图

习题与思考题

- 11.1 当你“编写”程序时是否会设计软件？软件设计和编码有什么不同？
- 11.2 如果软件设计不是程序（它肯定不是），那么它是什么？
- 11.3 如何评估软件设计的质量？
- 11.4 查看设计任务集，在任务集中的什么地方对质量进行评估？评估是如何完成的？如何达到 11.2.1 节中讨论的质量属性？

- 11.5 举三个数据抽象和能用来控制数据的过程抽象的例子。
- 11.6 用你自己的话描述软件体系结构。
- 11.7 为你每天都能遇到的东西（例如家用电器、汽车、设备）推荐一个设计模式，简要地描述该模式。
- 11.8 用你自己的语言描述关注点分离。分而治之的方法有时候不合适吗？这种情况对模块化的观点有多大的影响？
- 11.9 应在什么时候把模块设计实现为单块集成软件？如何实现？性能是实现单块集成软件的唯一理由吗？
- 11.10 讨论作为有效模块化属性的信息隐蔽概念和模块独立性概念之间的联系。
- 11.11 耦合性的概念如何与软件可移植性相关联？举例支持你的论述。
- 11.11. 应用“逐步求精方法”为下列一个或多个程序开发三种不同级别的过程抽象：（1）开发一个支票打印程序，给出金额总数，并按支票的常规要求给出大写金额数；（2）为某个超越方程迭代求解；（3）为操作系统开发一个简单的任务调度算法。
- 11.13 考虑需要实现汽车导航功能（GPS）、手持通信设备的软件。描述两个或三个要表示的横切关注点。讨论如何将其中一个关注点作为方面来表示。
- 11.14 “重构”意味着迭代地修改整个设计吗？如果不是，它意味着什么？
- 11.15 用你自己的语言描述什么是依赖倒置？
- 11.16 测试设计为什么很重要？
- 11.17 简要描述设计模型的四个元素。

扩展阅读与信息资源

Donald Norman 编写了三本书：《Emotional Design: We love(or hate) Everyday Things》（Basic Books, 2005）；《The Design of Everyday Things》（Doubleday, 1990）以及《The Psychology of Everyday Things》（HarperCollins, 1988）。这三本书已经成为设计学中的经典著作，任何人想设计人类使用的任何东西，都“必须”阅读这些著作。Adams 的著作（《Conceptual Blockbusting》，4th ed., Addison-Wesley, 2001）对那些希望拓宽思路的设计人员极为重要。最后，Polya 在其编写的一本经典著作（《How to Solve It》，2nd ed., Princeton University Press, 1988）中提供了通用的问题解决流程，在软件设计人员面对复杂问题时能够提供帮助。

Hanington 和 Martin（《Universal Methods of Design: 100 ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions》，Rockport, 2012 和《Universal Principles of Design: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design》，2nd ed., Rockport, 2010）讨论了通常的设计原则。

遵循同样的传统，Winograd 等人（《Bringing Design to Software》，Addison-Wesley, 1996）讨论了成功与不成功的软件设计及其理由。Wixon 和 Ramsey 编写了一本令人着迷的书（《Field Methods Casebook for Software Design》，Wiley, 1996），书中建议使用领域搜索方法（和人类学家所使用的那些方法非常类似）理解最终用户是如何工作的，然后设计满足用户需要的软件。Holtzblatt（《Rapid Contextual Design: A How-to Guide to Key Techniques for User-Center Design》，Morgan Kaufman, 2004）以及 Beyer 和 Holtzblatt（《Contextual Design: A Customer-Centered Approach to Systems Designs》，Academic Press, 1997）提供了软件设计的另一种视图，即将客户/用户集成到软件设计流程的各个方面。Bain（《Emergent Design》，Addison-Wesley, 2008）将模式、重构和测试驱动的开发方法结合到

有效的设计方法中。

Otero(《Software Engineering Design:Theory and Practice》,Auerbach,2012)、Venit 和 Drake(《Prelude to Programming: Concepts and Design》, 5th ed., Addison-Wesley, 2010)、Fox(《Introduction to software Engineering Design》, Addison-Wesley, 2006)以及 Zhu(《Software Design Methodology》, Butterworth-Heinemann, 2005)介绍了软件工程中设计的综合性处理。McConnell(《Code Complete》, 2nd ed.,Microsoft Press,2004)对高质量计算机软件的实践方面进行了精彩的论述。Robertson(《Simple Program Design》,5th ed., Course Technology,2006)介绍性地论述了软件设计,这对初学者很有帮助。Budgen(《Software Design》, 2nd ed., Addison-Wesley, 2004)介绍了大量流行的软件设计方法,并对它们进行了对比。Fowler和他的同事(《Refactoring: Improving the Design of Existing Code》, Addison-Wesley, 1999)讨论了软件设计增量优化的技术。Rosenberg 和 Stevens(《Use Case Driven Object Modeling with UML》, Apress, 2007)讨论了以用例为基础的面向对象的设计开发。

从 Free-man 和 Wasserman(《Software Design Techniques》, 4th ed., IEEE, 1983)编辑的文集中,可以一览软件设计的精彩发展历史。这本教程中转载了许多经典的论文,这些论文已经奠定了软件设计当前发展趋势的基础。Card 和 Glass(《Measuring Software Design Quality》, Prentice-Hall, 1990)从技术和管理两个角度介绍并思考了软件质量的度量。

网上有关于软件设计的大量信息资源。在 SEPA 网站 www.mhhe.com/pressman 上可以找到与软件设计以及设计工程相关的最新参考文献。

体系结构设计

要点浏览

概念: 体系结构设计表示建立计算机系统所需的数据结构和程序构件。它需要考虑系统采取的体系结构风格、系统组成构件的结构和属性以及系统中所有体系结构构件之间的相互关系。

人员: 尽管软件工程师能够设计数据和体系结构,但在构造大型复杂系统时,这项工作往往由专家来完成。数据库或者数据仓库设计者为系统创建数据体系结构。“系统架构师”根据软件需求分析中导出的需求选择合适的体系结构风格。

重要性: 没有图纸就不要试图盖房子,难道不是吗?同样,也不能通过勾画房子的管道布局而开始绘制房屋的蓝图。在开始考虑细节之前,需要关注“宏观”视图,即房子本身。这就是体系结构设

计需要做的事情——它提供“宏观”视图,并确保可以正确理解该视图。

步骤: 体系结构设计始于数据设计,然后导出系统体系结构的一个或多个表示。对可选的体系结构风格或模式进行分析,得出最适于客户需求和质量属性的结构。方案一旦选定,就需要使用体系结构设计方法对体系结构进行细化。

工作产品: 在体系结构设计过程中,要创建一个包括数据体系结构和程序结构的体系结构模型。此外,还需描述构件的属性以及关系(交互作用)。

质量保证措施: 在每个阶段,都要对软件设计的工作产品进行评审,以确保工作产品与需求之间以及工作产品彼此之间的清晰性、正确性、完整性和一致性。

设计通常被描述为一个多步过程,该过程从信息需求中综合出数据和程序结构的表示、接口特征和过程细节。Freeman[Fre80]扩展了该描述:

设计活动关注如何做出重要决策,而且往往是结构性的。设计和编程都关注抽象信息表示和处理顺序,但在详细程度上,两者俨然不同。设计是构建内聚的、良好规划的程序表示,它关注高层各部分之间的相互关系和低层所包括的逻辑操作。

正如第 11 章所提到的,设计是由信息驱动的。软件设计方法是通过仔细考虑分析模型的三个域而得到的。数据、功能和行为这三个域是创建软件设计的指南。

本章将介绍建立设计模型的数据和体系结构层的“内聚的、规划良好的表示”所需的方法。目标是提供一种导出体系结构设计的系统化方法,而体系结构设计是构建软件的初始蓝图。

关键概念

- 敏捷和体系结构
- 原型
- 体系结构决策
- 体系结构描述语言
- 体系结构描述
- 体系结构设计
- 体系结构类型
- 体系结构模式
- 体系结构风格
- 体系结构
- 体系结构一致性
- 检查
- 体系结构细化

12.1 软件体系结构

Shaw 和 Garlan[Sha96] 在他们划时代的著作中以如下方式讨论了软件的体系结构：

从第一个程序被划分成模块开始，软件系统就有了体系结构。同时，程序员已经开始负责模块间的交互和模块装配的全局属性。从历史的观点看，体系结构隐含了不同的内容——实现的偶然事件或先前的遗留系统。优秀的软件开发人员经常采用一个或者多个体系结构模式作为系统组织策略，但是他们只是非正式地使用这些模式，并且在最终系统中没有将这些模式清楚地体现出来。

如今，有效的软件体系结构及其明确的表示和设计已经成为软件工程领域的主导主题。

12.1.1 什么是体系结构

当你考虑建筑物的体系结构时，脑海中会出现很多不同的属性。在最简单的层面上，会考虑物理结构的整体形状。但在实际中，体系结构还包含更多的方面。它是各种不同建筑构件集成为一个有机整体的方式；是建筑融入所在环境并与相邻的其他建筑相互吻合的方式；是建筑满足既定目标和满足主人要求的程度；是对结构的一种美学观感（建筑的视觉效果），以及纹理、颜色和材料结合在一起创建外观和内部“居住环境”的方式；是很多微小的细节——灯具、地板类型、壁挂布置等的设计。总而言之，它是艺术。

体系结构也可以是其他的东西。它是“数以千计的或大或小的决定”[Tyr05]。其中一些决定是设计初期做出的，并可能会对所有其他设计行为产生深刻的影响。另外一些决定一直推迟到后来才做出，因此消除了过分限制性的制约因素，而这些制约因素可能会导致拙劣的体系结构风格。

但是，什么是软件体系结构呢？Bass、Clements 和 Kazman[Bas03] 对于这个难懂的概念给出了如下定义。

程序或计算系统的软件体系结构是指系统的一个或者多个结构，它包括软件构件、构件的外部可见属性以及它们之间的相互关系。

体系结构并非可运行的软件。确切地说，它是一种表达，使你能够：（1）对设计在满足既定需求方面的有效性进行分析；（2）在设计变更相对容易的阶段，考虑体系结构可能的选择方案；（3）降低与软件构建相关的风险。

该定义强调了“软件构件”在任意体系结构表示中的作用。在体系结构设计环境中，软件构件可能会像程序模块或者面向对象的类那样简单，但也可能扩充到包含数据库和能够完成客户与服务器网络配置的“中间件”。构件的属性是理解构件之间如何相互作用的必要特征。在体系结构层次上，不会详细说明内部属性（如算法的细节）。构件之间的关系可以像从一个模块对另一个模块进行过程调用那样简单，也可以像数据库访问协议那样复杂。

软件工程界（如 [Kaz03]）的一些成员对导出软件体系结构（称为“体系结构设计”）和导出软件设计这两种行为作了区分。正如之前版本的一位评论者指出：

“体系结构”和“设计”这两个术语之间有明显的不同。设计是体系结构的一个实例，类似于对象是类的实例一样。例如，考虑“客户-服务器”体系结构，我们可以使用 Java

引述 系统的体系结构是一个关于系统形式和结构的综合框架，包括系统构件和构件的整合。

Jerrold Grochow

关键点 软件体系结构必须对系统结构以及数据和过程构件相互协作的方式进行建模。

引述 体系结构忙中建，闲时悔。

Barry Boehm

平台 (Java EE) 或者 Microsoft 平台 (.NET 框架), 选择基于该体系结构的多种不同的实现方式设计一个网络中心软件系统。即使是同一个体系结构, 也可能产生多种基于该体系结构的设计。因此, 不能把“体系结构”和“设计”混为一谈。

尽管本书同意软件设计是特定软件体系结构的实例, 但元素和结构作为体系结构的一部分, 仍是每个设计的根本。设计开始于对体系结构的思考。

12.1.2 体系结构为什么重要

在一本关于软件体系结构的书中, Bass 和他的同事 [Bas03] 给出了软件体系结构之所以重要的三个关键原因:

- 软件体系结构提供了一种表示, 有助于对计算机系统开发感兴趣的所有利益相关者开展交流。
- 体系结构突出了早期的设计决策, 这些决策对随后所有的软件工作有深远的影响。
- 体系结构“构建了一个相对小的、易于理解的模型, 该模型描述了系统如何构成以及其构件如何一起工作” [Bas03]。

体系结构设计模型和包含在其中的体系结构模式都是可以传递的, 也就是说, 体系结构的类型、风格和模式 (12.2 ~ 12.6 节) 可以应用于其他系统的设计, 并且表示了一组抽象, 使得软件工程师能以可预见的方式描述体系结构。

网络资源 可以在 <http://www.ewi.ta.com/links/softwareArchitectureLinks.htm> 获得许多软件体系结构站点的可用链接。

关键点 体系结构模型提供了系统的 Gestalt 视图, 允许软件工程师在总体上审查它。

12.1.3 体系结构描述

对于体系结构这个词的意义, 每个人都会有一种理解。因为, 不同的参与者会从不同的角度理解体系结构, 这个角度是由不同的关注点驱动的。这就意味着体系结构描述实际上是一组体现系统不同视图的工作产品。

Smolander、Rossi 和 Purao [Smo08] 提出了多个比喻, 从不同的角度来说明同一体系结构, 以便不同的参与者能更好地理解软件体系结构这个术语。对于那些编写程序来实现系统的参与者来说, 他们更习惯于将其称为蓝图。开发人员将体系结构描述为一种传递准确信息的工具, 从体系结构分析师到设计师乃至生产系统构件的软件工程师之间都可以传递这种信息。语言则侧重于将其视为不同角色人群之间进行沟通的工具, 那些具有较高客户视野的参与者 (如管理者、市场专业人士等) 较偏向于这个角度。因为体系结构的描述为后续的协商奠定了基础, 特别是在确定系统边界方面, 故而它应该是简洁易懂的。

体系结构又可比喻为在对诸如成本、可用性、可维护性、性能等属性进行权衡取舍后做出的决策, 这些属性都会对系统设计产生重大影响。利益相关者 (如项目经理) 需基于体系结构的决策来分配项目资源和工作任务。这些决策可能会影响任务的排序和软件团队的组成。文献比喻对过往已经搭建完成的体系结构形成方案文档, 可以支持构建产品, 并且将产品设计思想传递给软件维护人员, 同样也支持关心构件和设计重用的项目相关人员。

软件体系的体系结构描述也必须展示出以上这些不同视角所组合的特征。Tyree 和 Akerman [Tyr05] 注意到了这一点, 他们写道:

开发人员想要对设计进行明确、果断的指导; 客户想要对必然发生的环境变化进行清晰的理解, 以及确保体系结构将满足他们的业务需要; 而体系结构设计师想要对体系结构的关

键方面进行清晰而深入的理解。

这里每一个“想要”的东西都反映了不同视点上的不同视角。

IEEE 计算机学会提出了 IEEE-Std-1471-2000, 即“密集型软件系统体系结构描述的推荐实践做法”(Recommended Practice for Architectural Description of Software-Intensive System)[IEE00], 目标如下:(1) 建立软件体系结构设计过程中使用的概念性框架和词汇表;(2) 提供表示体系结构描述的详细准则;(3) 鼓励良好的体系结构设计实践。体系结构描述(Architectural Description, AD)展示了多个视图, 每个视图都是“从一组参与者关注点的角度观察的整个系统的一种表示”。

12.1.4 体系结构决策

视图作为体系结构描述的一部分, 解决一个特定利益相关者的关注点。为了开发每个视图(和作为整体的体系结构描述), 系统体系结构设计师会考虑多种可选方案, 并最终就最能满足关注点的特定的体系结构特征做出决策。因此, 可以将体系结构决策本身看作体系结构的一种视图。通过理解做出体系结构决策的缘由可以深刻洞悉系统的结构以及系统与利益相关者关注点的一致性。

作为一名系统体系结构设计师, 可以使用下面推荐的模板记录每个主要的决策。通过记录, 设计师为他的工作提供了逻辑依据, 并且还可以建立历史记录, 该记录在需要进行设计修改时可能有用。

Grady Booch [Boo11a] 写道, 刚开始启动一项创新性的产品开发时, 软件工程师们往往会感觉到被迫一头扎进去, 他们搭建原材料、修复问题、改进现有的处理方式, 然后重复着这些过程。但是经过几轮反复之后, 他们意识到有必要将所选用的体系结构以及与之相关的决策清晰地表达出来。在构建一项新产品之前, 还无法预见到何为正确的选择。然而, 当开发者们对新的产品原型进行现场测试后, 觉得该项体系结构方案值得重复使用时, 那么针对此类产品的主导设计^①就开始显现了。如果不将工作中的成功与失败记录下来, 软件工程师们就很难决定何时需设计新的方案, 何时采用现有的体系结构方案。

信息栏 体系结构决策描述模板

每个主要的体系结构决策都可以被记录在案, 以便以后评审, 评审由想要理解已提出的体系结构描述的利益相关者进行。这里给出的是 Tyree 和 Ackerman [Tyr05] 提出模板的修改和缩略版本。

设计问题: 描述将要解决的体系结构设计问题。

解决方案: 陈述所选择的解决设计问题的方法。

分类: 指定问题和解决方案陈述的分类(例如, 数据设计、内容结构、构件结构、集成、简要说明)。

假设: 指出任何有助于制定决策的假设。

约束: 指定任何有助于制定决策的环境约束(例如, 技术标准、可用的模式、项目相关问题)。

候选方案: 简要描述所考虑的体系结构设计候选方案, 并描述为什么要摒弃这些方案。

^① 主导设计是指一种创新的软件体系结构或方法在市场上经过一段时间的成功适应和使用后而成为了工业标准。

争论: 陈述你为什么选择了这种解决方案而不是其他的候选方案。

意义: 指出制定决策对设计的影响。选择方案如何影响其他的体系结构设计问题?

解决方案会在某种程度上约束设计吗?

相关决策: 其他记录的决策和该决策有什么相关性?

相关关注点: 其他需求和该决策有什么相关性?

工作产品: 指出在体系结构描述中, 决策会在哪里体现出来。

注释: 参考可用来制定决策的其他团队的备忘录或文档。

12.2 体系结构类型

尽管体系结构设计的基本原则适用于所有类型的体系结构, 但对于需要构建的结构, 体系结构类型 (genre) 经常会规定特定的体系结构方法。在体系结构设计环境中, 类型隐含了在整个软件领域中的一个特定类别。在每种类别中, 会有很多的子类别。例如, 在建筑物类型中, 会有以下几种通用风格: 住宅房、单元楼、公寓、办公楼、工厂厂房、仓库等。在每一种通用风格中, 也会运用更多的具体风格 (12.3 节)。每种风格有一个结构, 可以用一组可预测模式进行描述。

Grady Booch 在他的《软件体系结构手册》[Boo08] 的改进版本中, 提出了以下几种软件系统的体系结构类型, 包括: 人工智能、通信、设备、金融、游戏、工业、法律、医疗、军事、操作系统、运输、实用程序以及许多其他类型。

12.3 体系结构风格

当建筑师用短语“殖民式中厅”(center hall colonial) 来描述某座房子时, 大多数熟悉美国房子的人能够产生一种整体画面, 即房子看起来是什么样子以及主平面图看起来是什么样子。建筑师使用体系结构风格作为描述手段, 将该房子和其他风格 (例如, A 框架、砖房、鲑鱼角式等) 的房子区分开来。但更重要的是, 体系结构风格也是建筑的样板。必须进一步规定房子的细节, 具体说明它的最终尺寸, 进一步给出定制的特征, 确定建筑材料等。实际上是建筑风格——“殖民式中厅”——指导了建筑师的工作。

基于计算机系统构造的软件也展示了众多体系结构风格中的一种。每种风格描述一种系统类别, 包括: (1) 完成系统需要的某种功能的一组构件 (例如, 数据库、计算模块); (2) 能使构件间实现“通信、合作和协调”的一组连接件; (3) 定义构件如何集成为系统的约束; (4) 语义模型, 能使设计者通过分析系统组成成分的已知属性来理解系统的整体性质 [Bas03]。

体系结构风格就是施加在整个系统设计上的一种变换, 目的是为系统的所有构件建立一个结构。在对已有体系结构再工程时, 体系结构风格的强制采用会导致软件结构的根本性改变, 包括对构件功能的再分配 [Bos00]。

与体系结构风格一样, 体系结构模式也对体系结构设计施加一种变换。然而, 体系结构

关键点 许多不同的体系结构风格可以用于一种特定的类型 (也称为应用领域)。

引述 每位艺术家的思想背后都蕴涵着某种体系结构的模式或者类型。

G. K. Cheserton

提问 什么是体系结构风格?

模式与体系结构风格在许多基本方面存在不同：(1) 模式涉及的范围要小一些，它更多集中在体系结构的某一方面而不是体系结构的整体；(2) 模式在体系结构上施加规则，描述了软件是如何在基础设施层次（例如并发）[Bos00] 上处理某些功能性方面的问题；(3) 体系结构模式（12.3.2 节）倾向于在体系结构环境中处理特定的行为问题（例如，实时应用系统如何处理同步和中断）。模式可以与体系结构风格结合起来建立整个系统结构的外形。

12.3.1 体系结构风格的简单分类

在过去的 60 年中，尽管已经创建了数百万的计算机系统，但绝大多数都可以归为少数的几种体系结构风格之一。

以数据为中心的体系结构。数据存储（如文件或数据库）位于这种体系结构的中心，其他构件会经常访问该数据存储，并对存储中的数据进行更新、增加、删除或者修改。图 12-1 描述了一种典型的以数据为中心的体系结构风格，其中，客户软件访问中心存储库。在某些情况下，数据存储库是被动的，也就是说，客户软件独立于数据的任何变化或其他客户软件的动作而访问数据。该方法的一个变种是将中心存储库变换成“黑板”，当客户感兴趣的数据发生变化时，它将通知客户软件。

以数据为中心的体系结构促进了可集成性（integrability）[Bas03]，也就是说，现有的构件可以被修改，而且新的客户构件可以加入到体系结构中，而无需考虑其他的客户（因为客户构件是独立运作的）。另外，数据可以在客户间通过“黑板”机制传送（即黑板构件负责协调信息在客户间的传递），客户构件独立地执行过程。

数据流体系结构。当输入数据经过一系列计算构件和操作构件的变换形成输出数据时，可以应用这种体系结构。管道-过滤器模式（图 12-2）拥有一组称为过滤器的构件，这些构件通过管道连接，管道将数据从一个构件传送到下一个构件。每个过滤器独立于其上游和下游的构件而工作，过滤器设计要针对某种形式的数据输入，并且产生某种特定形式的数据输出（到下一个过滤器）。然而，过滤器没有必要了解与之相邻的其他过滤器的工作。

网络资源 基于属性的体系结构风格（ABAS）可用作软件体系结构的构造块。可从 www.sei.cmu.edu/architecture/abas.html 获得相关信息。

引述 设计模式和风格的使用在工程学科中是非常普遍的。
Mary Shaw,
David Garlan

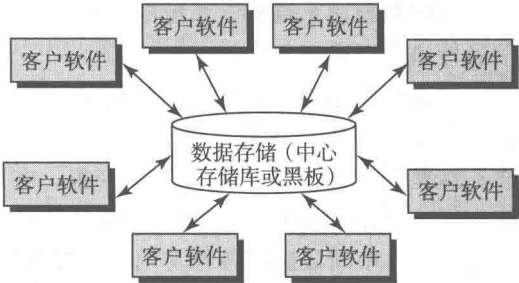


图 12-1 以数据为中心的体系结构

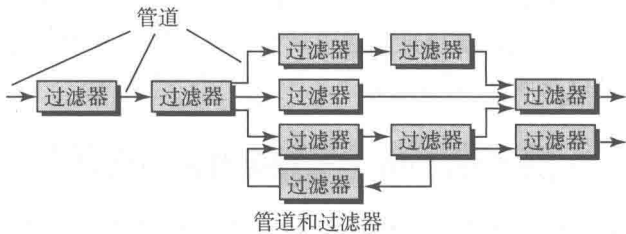


图 12-2 数据流体系结构

如果数据流退化成单线变换，则称为批处理序列（batch sequential）。这种结构接收一批

数据，然后应用一系列连续的构件（过滤器）完成变换。

调用和返回体系结构。该体系结构风格能够设计出一个相对易于修改和扩展的程序结构。在此类体系结构中，存在几种子风格 [Bas03]：

- **主程序 / 子程序体系结构。**这种传统的程序结构将功能分解为一个控制层次，其中“主”程序调用一组程序构件，这些程序构件又去调用其他构件。图 12-3 描述了该类型的体系结构。
- **远程过程调用体系结构。**主程序 / 子程序体系结构的构件分布在网络中的多台计算机上。

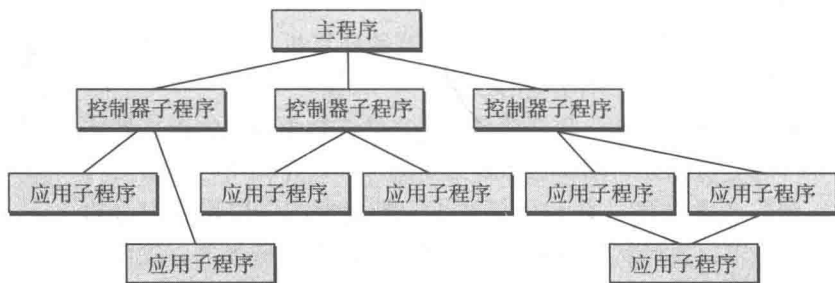


图 12-3 主程序 / 子程序体系结构

面向对象体系结构。系统的构件封装了数据和必须用于控制该数据的操作，构件间通过信息传递进行通信与合作。

层次体系结构。层次体系结构的基本结构如图 12-4 所示。其中定义了一系列不同的层次，每个层次各自完成操作，这些操作逐渐接近机器的指令集。在外层，构件完成建立用户界面的操作；在内层，构件完成建立操作系统接口的操作；中间层提供各种实用工具服务和应用软件功能。

以上描述的体系结构风格仅仅是可用风格中的一小部分^①。一旦需求工程揭示了待构建系统的特征和约束，就可以选择最适合这些特征和约束的体系结构风格或风格的组合。在很多情况下，会有多种模式是适合的，需要对可选的体系结构风格进行设计和评估。例如，在很多数据库应用中，层次体系结构（适合大多数系统）可以与以数据为中心的体系结构结合起来使用。

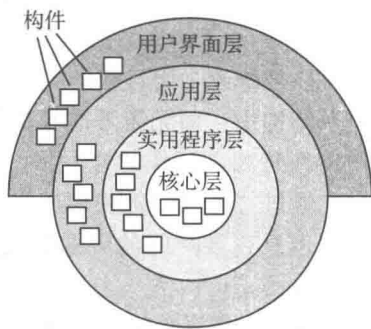


图 12-4 层次体系结构

确定一个合适的体系结构风格殊为不易，Buschman [Bus10a] 提出的两种补充思想可以对此提供一些指导。问题帧是指在不考虑专业领域知识或程序实现方案细节的前提下，描述反复出现的问题的特征。领域驱动设计则建议软件设计应该反映出你的应用（第 7 章）试图解决的业务问题的领域以及该领域的逻辑。

SafeHome 选择体系结构风格

[场景] Jamie 的房间，设计建模还在继续
进行。

[人物] Jamie 和 Ed，SafeHome 软件工
程团队的成员。

^① 体系结构风格与模式的更详细讨论参见 [Roz11]、[Tay09]、[Bus07]、[Gor06] 或 [Bas03]。

[对话]

Ed (皱着眉): 我们已经使用 UML 对安全功能进行了建模……类、关系等都是其中的基本材料, 所以我觉得面向对象体系结构^①应该是合适的方案。

Jamie: 但是……

Ed: 但是……我对于面向对象体系结构理解起来有些困难, 我比较熟悉调用和返回体系结构——一种传统的过程层次。但是面向对象……我不了解, 它看起来属于无组织的一类。

Jamie (微笑): 无组织?

Ed: 是的……我的意思是说我不能想象出实际的结构, 在设计空间中只有设计类。

Jamie: 哦, 那不对。存在类的层次……想想我们为 FloorPlan 对象设计的层次(聚集)(图 11-3)。面向对象的体系结构是结构与类之间相互连接的组合, 你知道, 类之间的相互连接即相互协作。我们可以通过描述详细的类结构、属性、操作和类之间的消息来体现它。

Ed: 我打算花一个小时制定一个调用和返回体系结构, 然后我再考虑面向对象体系结构。

Jamie: Doug 对此不会有什么问题, 他说我们应该考虑其他方案。顺便说一句, 这两种体系结构彼此结合是绝对没有问题的。

Ed: 好, 我知道了。

一个问题帧是指对同一类问题进行概括以用来解决所遇到的问题, 有五种通常与体系结构风格相关联的基础问题帧: 简单工作片段(工具)、需求行为(以数据为中心)、指令行为(指令处理器)、信息显示(观察者)以及转换(管道和过滤器变换)。

现实世界的问题常常与多个问题帧相关, 相应地, 一种体系结构模型有可能是不同问题帧的组合。例如, 在 WebApp 设计中使用的模型-视图-控制器(MVC)体系结构可视为两类问题帧(指令行为和信息显示)的组合。在 MVC 中, 最终用户由浏览器窗口发送指令至指令处理器(控制器), 控制器负责对内容(模型)的访问, 并指示信息生成模型(视图)将其转换至浏览器显示。

领域建模会影响到体系结构风格的选择, 领域对象的核心特性尤其如此。表示物理对象的领域对象(如传感器或驱动器)应该与表示逻辑对象的领域对象(如任务调度、工作流)区别开来。物理对象必须严格遵守约束, 如连接限制或消耗资源的使用。逻辑对象可以有一些可被取消或解除的温和的实时行为。层次体系结构对领域驱动设计的支持度最好。[Eva04]

12.3.2 体系结构模式

开发需求模型时将会注意到软件必须解决许多问题, 这些问题很广泛, 跨越了整个应用领域。例如, 对于几乎每一个电子商务应用系统, 其需求模型都会遇到下述问题: 我们如何给各方面的客户提供不同类型的产品, 并且允许这些客户在线购买我们的产品?

需求模型也定义了必须对上述问题进行回答的环境。例如, 面向顾客销售高尔夫设备的电子商务和面向媒体和大中型公司销售高价工业设备的电子商务, 它们的

引述 也许这是在地下室, 让我们上楼去检查。

M. C. Escher

① 可能有一种争议: SafeHome 体系结构的层次应比指明的层次更高。SafeHome 有许多不同的子系统——住宅监控功能、公司的监控点以及运行在房主 PC 上的子系统。在子系统中, 并行过程(即那些监控传感器)和事件处理非常普遍。在产品工程过程中可以做出该层次上的某些体系结构决策, 但在软件工程中, 体系结构设计可能要考虑这些问题。

营运环境完全不同。另外，一系列限制和约束可能会影响到需要解决问题的处理方式。

体系结构模式在特定环境和一系列限制与约束下处理特定的应用问题。模式提出了能够作为体系结构设计基础的体系结构解决方案。

本章前面的部分曾提到过，大多数应用系统都符合特定领域或特定类型，适合于这种类型的风格有一种或者多种。例如，一个应用系统的整体体系结构风格可能是“调用和返回”或者“面向对象”型，但在那种风格中，你会遇到一系列常见问题，这些问题最好是用具体的体系结构模式来处理。

12.3.3 组织和求精

由于设计过程经常会留下许多种可供选择的体系结构方案，因此建立一组用于评估所导出的体系结构设计的设计标准是非常重要的。下面的问题 [Bas03] 有助于更深入地了解体系结构风格：

控制。在体系结构中如何管理控制？是否存在清楚的控制层次？如果存在，构件在控制层次中有什么作用？构件如何在系统中传递控制？构件间如何共享控制？控制的拓扑结构（即控制呈现的几何形状）如何？控制是否同步或者构件操作是否异步？

提问 如何评估导出的体系结构风格？

数据。构件间如何进行数据通信？数据流是否连续地传递给系统，或数据对象是否零散地传递给系统？数据传递的模式是什么（数据是从一个构件传递到另一个构件，还是数据被系统中的构件全局共享）？是否存在数据构件（如黑板或中心存储库）？如果存在，它们的作用是什么？功能构件如何和数据构件进行交互？数据构件是被动的还是主动的（数据构件是否主动地和系统中的其他构件进行交互）？系统中的数据和控制如何进行交互？

这些问题有助于设计者对设计质量进行早期评估，也为更详细的体系结构分析奠定了基础。

演化过程模型（第4章）已变得非常流行，这意味着软件体系结构可能需要随着每次产品增量的计划与实施而演变发展。在第11章中我们将此过程描述为重构，即在不改变产品外在行为的情况下对其内部结构进行改进。

12.4 体系结构考虑要素

Buschmann 和 Henny [Bus10b, Bus10c] 提出了几个考虑要素，指导软件工程师在体系结构设计时做出决策。

- **经济性**——许多软件体系结构深受不必要的复杂性所害，它们充斥着不必要的产品特色或无用的需求（如无目的的可重用性）。最好的软件应该是整洁的并依赖抽象化以减少无用的细节。
- **易见性**——设计模型建立后，对于那些随后将验证这些模型的软件工程师而言，体系结构的决策及其依据应该是显而易见的。如果重要的设计和专业领域概念与随后的设计和开发人员没有进行有效沟通，所产生的设计模型往往是晦涩难懂的。
- **隔离性**——不产生隐藏依赖的关注点分离是非常理想的设计思想（第11章），有时我们将此称为隔离性。适当的隔离会产生模块化的设计，但过分的隔离又会导致碎片

提问 开发软件体系结构时应该考虑哪些要素呢？

化和易见性的丧失。诸如领域驱动的设计方法可以协助确定哪些应当分离，哪些应当处理为连贯的单元。

- **对称性**——体系结构的对称性意味着它的属性是均衡一致的，对称的设计更易于理解、领悟和沟通。比如，设想一个客户账户的对象，其生命周期可被软件体系结构直接模式化为同时提供 `Open()` 和 `Close()` 方法。体系结构的对称性可同时包括结构上的对称性和行为上的对称性。
- **应急性**——紧急的、自组织的行为和控制常常是创建可扩展的、经济高效的软件体系结构的关键。比如：许多实时性的软件应用是由事件驱动的，定义系统行为的事件序列和它们的持续时间确定了应急响应的质量，很难预先规划好事件所有可能的序列，相反，系统体系结构设计师应构建一个灵活系统，能够适应这类突发事件的出现。

以上这些考虑要素并非独立存在，他们之间既相互作用又相互调节。例如，隔离性可因经济性而加强或减轻，隔离性也可平衡易见性。

软件产品的体系结构描述在实现它的源代码中并非显而易见。相应地，随着源代码的不断修改（如软件维护活动），软件体系结构也将逐渐被侵蚀。对设计者而言，如何对体系结构信息进行适当的抽象是一项挑战。这些抽象可潜在地提高源代码的结构化程度，从而改善可读性和可维护性 [Bro10b]。

SafeHome 评估体系结构决策

[场景] Jamie 的房间，设计建模还在继续进行。

[人物] Jamie 和 Ed，SafeHome 软件工程师团队的成员。

[对话]

Ed: 我完成了安全功能的调用返回体系结构模型。

Jamie: 太好了，你觉得达到了我们的要求吗？

Ed: 它没有任何不必要的功能，看起来很简洁。

Jamie: 易见性如何？

Ed: 还不错，我觉得这个模型用来实现这个产品的安全需求毫无问题。

Jamie: 我相信你已经理解了该体系结构，但你可能不负责该部分的程序开发，我有一点担心隔离性，作为面向对象的设计，这个模型可能还不够模块化。

Ed: 可能是这样，但是当我们创建 SafeHome 的 Web 版的时候，我担心会限制代

码重用的能力。

Jamie: 对称性如何呢？

Ed: 也还可以，我比较难以评估这一块，因为对我而言，在安全功能中仅有的存在对称性的地方就是增加和删除 PIN 信息。

Jamie: 那么当我们在 Web 版中增加远程安全模块时会变得更复杂了。

Ed: 我想是这样。

(考虑到体系结构的异议，他们同时陷入了沉思。)

Jamie: SafeHome 是一个实时系统，所以状态的转换和事件的次序比较难以预见。

Ed: 是的，不过这个系统的应急响应可以用一个有限状态模型来处理。

Jamie: 如何处理呢？

Ed: 该模型可以基于调用返回结构来实现，在很多编程语言中，中断都比较容易处理。

Jamie: 你认为我们有必要对最初考虑的面向对象体系结构进行同样的分析吗？

Ed: 好主意,一旦开始实施后,体系结构就很难变动了。

Jamie: 在这些体系结构之上,再审视一下

除安全之外的其他非功能性需求也很重要,以确保它们都被通盘考虑了。

Ed: 对。

12.5 体系结构决策

与系统体系结构相关的决策记录了关键的设计问题以及所选用的体系结构方案背后的原理。一些决策包括软件系统组织、结构元素的选取和它们之间的协作意向所定义的接口,以及这些元素组合而成的越来越大的子系统 [Kru09]。另外,还需要进行体系结构模式、应用技术、中间件及编程语言的选择。体系结构决策的成果会影响到系统的非功能特性及其众多的质量属性 [Zim11],这些成果能够以开发者笔记的形式记录下来。这些笔记记录了关键性的设计决策以及支撑它们的理由,为新的项目团队成员提供了参考,也可以作为经验学习的知识库。

通常,软件体系结构实践的着眼点是呈现和记录不同类别参与者的需要。然而,定义一种决策视图来贯穿传统的体系结构表现手段中所包含的多种信息视角是有可能的。这种决策视图捕获了体系结构设计决策以及做出决策的依据。

面向服务的体系结构决策 (SOAD)^①建模 [Zim11] 是一种知识管理框架,它以一种可以指导未来开发活动的方式,为捕获体系结构决策的依赖提供了支持。

将一种体系结构风格应用于某一特定的应用类型时,指导模型包含了此体系结构决策所要求的相关知识。该模型基于已完成项目中获取的体系结构信息而建立,并且此类项目采用了前述的体系结构风格。指导模型记载了设计问题存在的地方、应当做出体系结构决策的地方,以及从潜在可选方案中作挑选时应当考虑的质量属性。潜在的可选方案(以及各自的利弊)是从之前的软件应用中总结出来的,以辅助体系结构设计师做出最好的决策。

决策模型记录了所需要的体系结构决策、在过往的项目中实际做出的决策以及支持这些决策的理由。指导模型为体系结构决策模型提供了一种可裁剪的步骤,它允许体系结构设计师删除不相关的议题、扩展重要的议题或是添加新的议题。一个决策模型可以利用多个指导模型,并在项目完成后向指导模型提供反馈。这种反馈可从项目完成后的经验总结评审中发掘出来。

12.6 体系结构设计

在体系结构设计开始的时候,应先建立相应的环境。为达成此目标,应该定义与软件交互的外部实体(其他系统、设备、人)和交互的特性。这些信息一般可以从需求模型中获得。一旦建立了软件的环境模型,并且描述出所有的外部软件接口,就可以确定体系结构原型集。

原型是表示系统行为元素的一种抽象(类似于类)。这个原型集提供了一个抽象集,如果要使系统结构化,就必须要对这些原型进行结构化建模,但原型本身并不提供足够的实施细节。因此,设计人员通过定义和细

引述 医生可以文过饰非,但是建筑师只能建议他的客户牵萝补屋。

Frank Lloyd Wright

提问 什么是原型?

^① SOAD 类似于体系结构模式的应用,更多的资料可由此链接中获取: <http://soadecisions.org/soad.htm>。

化实施每个原型的软件构件来指定系统的结构。这个过程持续迭代，直到获得一个完善的体系结构。

当软件工程师建立有真实意义的体系结构图时，应先自问并回答一系列问题 [Boo11b]。该图是否能显示系统对输入或事件的响应？哪些部分可以可视化地表达出来，以突出显示风险领域？如何将隐藏的系统设计模式展现给其他开发者？可否以多个视角展现最佳路径以分解系统的特定部分？设计中的各种权衡取舍能否有意义地展现出来？如果一个软件体系结构的图示可以回答以上这些问题，那么对于使用它的软件工程师而言将是很有价值的。

12.6.1 系统环境的表示

在体系结构设计层，软件体系结构设计师用体系结构环境图（Architectural Context Diagram, ACD）对软件与其外围实体的交互方式进行建模。图 12-5 给出了体系结构环境图的一般结构。

根据图中所示，与目标系统（为目标系统所开发的体系结构设计）交互的系统可以表示为：

- 上级系统——这些系统把目标系统作为某些高层处理方案的一部分。
- 下级系统——这些系统被目标系统使用，并为完成目标系统的功能提供必要的数据和处理。
- 同级系统——这些系统在对等的基础上相互作用（即信息或者由同级系统和目标系统产生，或者被目标系统和同级系统使用）。
- 参与者——通过产生和消耗必要处理所需的信息，实现与目标系统交互的实体（人、设备）。

每个外部实体都通过某一接口（带阴影的小矩形）与目标系统进行通信。

为了说明 ACD 的使用，再来考虑 SafeHome 产品的住宅安全功能。整个 SafeHome 产品的控制器和基于因特网的系统对于安全功能来说都处于上一级，在图 12-6 中它们在上方。监视功能是一个同级系统，并且在以后的产品版本中还要使用住宅安全功能（或被住宅安全功能使用）。房主和控制面板都是参与者，它们既是安全软件所用信息的生产者，又是安全软件所供信息的使用者。最后，传感器为安全软件所使用，并且在图中显示为下一级。

作为体系结构设计的一部分，必须说明图 12-6 中每个接口的细节。目标系统所有的流入和流出数据必须在这个阶段标识出来。

12.6.2 定义原型

原型（archetype）是表示核心抽象的类或模式，该抽象对于目标系统体系结构的设计非常关键。通常，即使设计相对复杂的系统，也只需要相对较小的原型集合。目标系统的体系

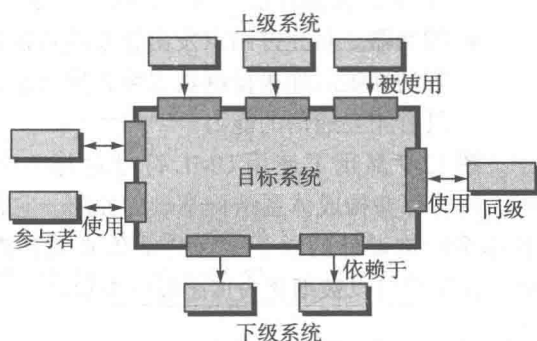


图 12-5 体系结构环境图 [Bos00]

提问 系统之间如何交互？

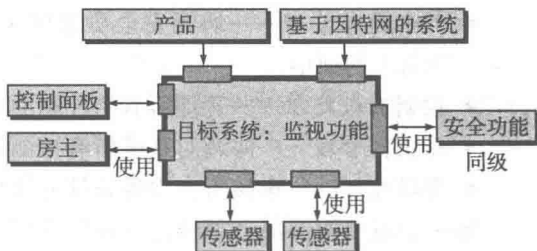


图 12-6 SafeHome 安全功能的体系结构环境图

结构由这些原型组成，这些原型表示体系结构中稳定的元素，但可以基于系统行为以多种不同的方式对这些元素进行实例化。

很多情况下，可以通过检验作为需求模型一部分的分析类来导出原型。继续关于 SafeHome 住宅安全功能的讨论，可能会定义下面的原型：

- **结点**。表示住宅安全功能的输入和输出元素的内聚集合，例如，结点可能由如下元素构成：（1）各种传感器；（2）多种警报（输出）指示器。
- **探测器**。对所有为目标系统提供信息的传感设备的抽象。
- **指示器**。表示所有指示警报条件发生的报警机械装置（例如，警报汽笛、闪灯、响铃）的抽象。
- **控制器**。对允许结点发出警报或者撤销警报的机械装置的抽象。如果控制器安装在网络上，那么它们应该具有相互通信的能力。

图 12-7 显示了使用 UML 符号对每一个原型的描述结果。回想那些构成体系结构基础的原型，它们是抽象的，随着体系结构设计的进行，这些抽象必须被进一步求精。例如，探测器可以被细化为传感器的类层次。

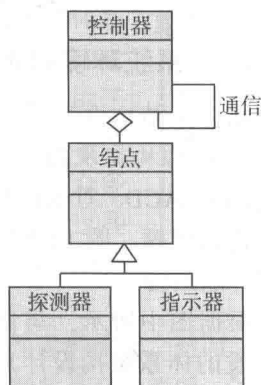


图 12-7 SafeHome 安全功能原型的 UML 关系 [Bos00]

12.6.3 将体系结构细化为构件

在将软件体系结构细化为构件时，系统的结构就开始显现了。但是，如何选择这些构件呢？为了回答这个问题，先从需求模型所描述的类开始^①。这些分析类表示软件体系结构中必需处理的应用（业务）领域的实体。因此，应用领域是构件导出和细化的一个源泉。另一个源泉是基础设施域。体系结构必须提供很多基础设施构件，使应用构件能够运作，但是这些基础设施构件与应用领域没有业务联系。例如，内存管理构件、通信构件、数据库构件和任务管理构件经常集成到软件体系结构中。

体系结构环境图（12.6.1 节）描述的接口隐含着一个或者多个特定的构件，这些构件处理经过接口的数据。在某些情况下（如图形用户界面），需要设计具有许多构件的、完整的子系统体系结构。

继续 SafeHome 住宅安全功能的例子，可以定义完成下列功能的顶层构件集合：

- **外部通信管理**——协调安全功能与外部实体（例如，基于 Internet 的系统与外部报警通知）的通信。
- **控制面板处理**——管理所有的控制面板功能。
- **探测器管理**——协调对系统所有探测器的访问。
- **警报处理**——审核所有报警条件并执行相应动作。

每一个顶层构件都必须经过反复的迭代细化，然后在总的 SafeHome 体系结构中进行定位。每个构件都需要定义设计类（包含相应的属性和操作）。然而，重要的是，在进行构件级设计之前，不要说明所有属性和操作的设计细节（第 13 章）。

关键点 原型是体系结构设计的抽象构造块。

引述 软件系统的结构提供一种“生态环境”，代码在这个环境里诞生、成长和消亡。一个设计良好的“生态环境”允许软件系统所需要的所有构件成功进化。

R. Pattis

① 如果选择了常规方法（非面向对象），那么构件可能从子程序调用层中导出（图 12-3）。

图 12-8 描述了整体体系结构（由 UML 构件图表示）。当事务从处理 SafeHome 的 GUI（图形用户界面）和 Internet 接口的构件进入时，它们就被外部通信管理获取。该信息由用于选择合适产品功能（安全功能）的 SafeHome 执行者构件来管理。控制面板处理构件通过与房主交互来实现安全功能的安装或解除。探测器管理构件定时查询传感器以检测报警条件，一旦检测到警报，警报处理构件将产生相应的输出。

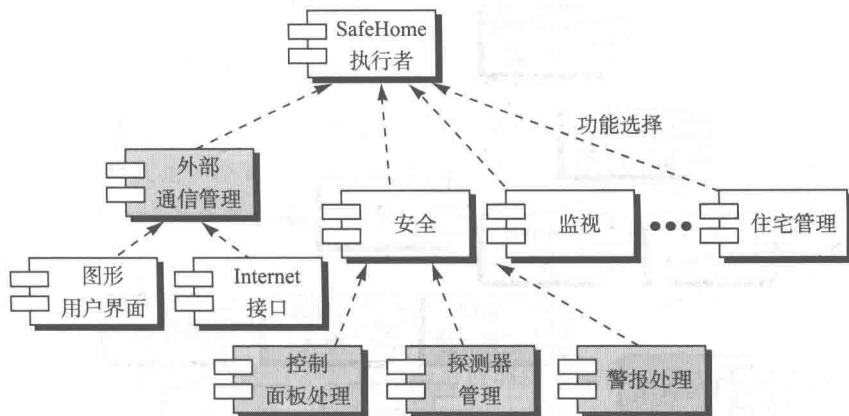


图 12-8 带有顶层构件的 SafeHome 整体体系结构

12.6.4 描述系统实例

到目前为止，所建模的体系结构设计依然处于比较高的层次。系统环境已经被描述，问题域中重要抽象的原型已经被定义，系统的整体结构已经显现，并且主要的软件构件也都确定了。然而，更进一步的细化（回想一下所有的设计都是迭代的）仍然是必要的。

为了进行体系结构求精，需要开发体系结构的实际用例。这样做的用意是将体系结构应用到特定问题上，证明结构和构件都是合理的。

图 12-9 描述的是安全系统 SafeHome 体系结构的一个实例。图 12-8 中显示的构件被进一步细化以显示更多的细节。例如，探测器管理构件与调度器基础设施构件相互作用，此基础设施构件实现安全系统中使用的每个传感器对象的定时查询。图 12-8 中显示的每个构件都作了类似的细化。

软件工具 体系结构设计

[目标] 体系结构设计工具通过描述构件接口、依赖与联系以及交互作用来建立整体软件结构模型。

[机制] 工具采用的机制多种多样。在大多数情况下，体系结构的设计能力是分析和设计建模自动化工具的一部分功能。

[代表性工具]^①

- Adalon。由 Synthis 公司（www.synthis.com）开发，是一种专用设计工具，用于设计和构建特定的基于 Web 构件的体系结构。
- ObjectiF。由 microTOOL GmbH（www.microtool.com）开发，是一种通用设计工具，用于设计和构建基于 Web 构件的体系结构。

① 这里提到的工具只是此类工具的例子，并不代表本书支持使用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

microtool.de/objectiF/en/) 开发, 是一个基于 UML 的设计工具, 它导出的体系结构 (例如 Coldfusion、J2EE 和 Fusebox 等) 与基于构件的软件工程 (第 13 章) 相符。

- Rational Rose。由 Rational (<http://www-01.ibm.com/software/rational/>) 开发, 是基于 UML 的设计工具, 它支持体系结构设计中的所有方面。

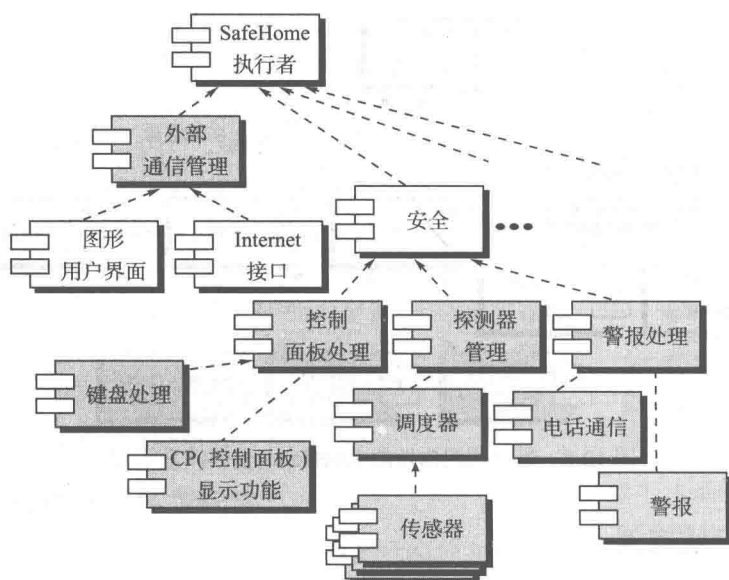


图 12-9 构件细化的安全功能实例

12.6.5 WebApp 的体系结构设计

WebApp 是典型的使用多层次体系结构来构造的客户端-服务器应用软件, 包括用户界面或表现层、一个基于一组业务规则来指导与客户端浏览器进行信息交互的控制器, 以及可以包含 WebApp 的业务规则的内容层或模型层。

WebApp 的用户界面是围绕着运行在客户端 (通常为个人计算机或移动设备) 上的浏览器的特性来设计的。数据层位于服务器。业务规则既可以使用基于服务器的脚本语言 (如 PHP) 实现, 也可以使用基于客户端的脚本语言 (如 Javascript) 来实现。体系结构设计师应根据安全性和可用性的需求来分配客户端和服务端的功能。

WebApp 的体系结构设计也受客户端所访问的内容结构 (线性或非线性的) 的影响。WebApp 的体系结构构件 (Web 页) 被设计为可控的, 以传递给系统的其他构件, 允许非常灵活的导航结构。媒体以及其他内容资源的物理位置也会对软件工程师确定体系结构产生影响。

13.6.6 移动 App 的体系结构设计

移动 App 是典型的使用多层体系结构来构造的系统, 包括用户界面层、业务层以及数据层。对于移动 App, 你可以选择建立一个基于 Web 的瘦客户端或是一个富客户端。对于瘦客户端, 只有用户界面位于移动设备上, 业务层和数据层都在服务端。而对于富客户端, 所有三层都位于移动设备本身。

每一部移动设备都因它们的物理特性 (如屏幕大小、输入设备)、软件 (如操作系统、所

支持的语言)以及硬件(如内存、网络连接)的不同而有所不同。每一种特性都勾画出了可供选择的体系结构的方向。Meier 和他的同事 [Mei09] 建议了许多考虑要素,这些要素可能影响移动 App 体系结构的设计:(1)将要建立的 Web 客户类型(瘦或富客户端);(2)所支持的设备种类(如智能手机、平板电脑);(3)连接程度需求(偶尔的还是持久的);(4)带宽需求;(5)移动平台的限制;(6)重用和可维护性的程度也是重要的;(7)设备资源的限制(如电池寿命、内存大小、处理器速度)。

12.7 评估候选的体系结构设计

Clements 和他的同事 [Cle03] 在他们关于软件体系结构进化的著作中声称:

说穿了,体系结构就是一个赌注,将注下在系统的成功上。如果你已经提前把赌注放在赢家,而不是等到系统快完成的时候才知道它是否满足需求,这样不是很好吗?如果你准备购买一个系统或者为系统开发付费,难道你不愿意有某种保证,使得系统不偏离正确的道路?如果你自己就是系统架构师,难道你不愿意有一种好方法来验证你的直觉和经验,从而知道所依赖的设计是有基础的,夜里能睡得踏实?

回答这些问题确实有价值。设计会导致多种可选的体系结构,其中每一种可选体系结构都需要进行评估,以确定哪种体系结构最适合要解决的问题。在下面几节中,我们提出两种不同的候选体系结构设计的评估方法。第一种方法使用迭代方法评估设计权衡,第二种方法运用伪定量技术来评估设计质量。

软件工程研究所(SEI)开发了一种体系结构权衡分析方法(Architecture Trade-off Analysis Method, ATAM) [Kaz98],该方法建立了一个迭代的软件体系结构评估过程。下面的设计分析活动是迭代进行的:

1. 收集场景。开发一组用例(第7章和第8章),从用户角度描述系统。
2. 引出需求、约束和环境描述。这些信息作为需求工程的一部分得到确定,并用来确保所有利益相关者的关注点都会被处理。
3. 描述那些已经被选择用于解决场景和需求的体系结构风格或模式。体系结构风格应该使用下面任意一种体系结构视图来描述:
 - 模块视图用于分析带有构件的工作任务以及信息隐蔽的程度。
 - 过程视图用于分析系统性能。
 - 数据流视图用于分析体系结构满足功能需求的程度。
4. 通过单独考虑每个属性来评估质量属性。用于分析的质量属性个数是评审可用时间和质量属性与现存系统相关程度的函数。体系结构设计评估的质量属性包括可靠性、性能、安全性、可维护性、灵活性、可测试性、可移植性、可复用性和互操作性。
5. 针对特定的体系结构风格,确定质量属性对各种体系结构属性的敏感性。这可以通过对体系结构做小的变更并确定某质量属性(如性能)对该变更的敏感性来完成。受体系结构变更影响很大的属性称为敏感点。
6. 使用在第5步进行的敏感性分析来鉴定候选体系结构(在第3步开发的)。SEI 用如下方式描述该方法 [Kaz98]:

一旦确定了体系结构敏感点,寻找这种权衡点就简单了,只需识别出对多个属性敏感的体系结构元素。例如,客户-服务器体系结构的性能可能对服务器数量是高度敏感的(在一定范围内,增加服务器的数量可使性能提高)……那么,服务器数量就是该体系结构的一个

权衡点。

这6个步骤描述了首次 ATAM 迭代。基于第5步和第6步的结果,某些候选体系结构可能被删除,剩余的一个或多个体系结构可能被修改和进一步细化,然后,再次应用 ATAM 步骤^①。

SafeHome 体系结构评估

[场景] Doug Miller 的办公室,体系结构设计建模正在进行。

[人物] Vinod、Jamie 和 Ed, SafeHome 软件工程团队成员; Doug Miller, 软件工程团队经理。

[对话]

Doug: 我知道大家为 SafeHome 这个产品设计了两个不同的体系结构,这是一件好事。我的问题是,我们该如何选择最好的体系结构呢?

Ed: 我正在设计一个调用和返回风格的体系结构,然后 Jamie 或我将设计一个面向对象的体系结构。

Doug: 好的,但是我们如何选择呢?

Jamie: 我在高年级时学习了一门计算机科学课程,我记得有很多选取办法。

Vinod: 是有一些,但是它们都太理论化了。听着,我认为我们可以自己评估,并使用用例和场景来选择正确的体系结构。

Doug: 这不是一回事吗?

Vinod: 当谈论有关体系结构评估的时候,它们不是一回事。我们已经有了一个完整的用例集合,因此,我们将每个用例都应

用到这两个体系结构中,查看系统的反应,即在用例环境中构件和连接件是如何工作的。

Ed: 这是个好主意!可以确保我们不遗漏任何东西。

Vinod: 当然,它还能告诉我们体系结构设计是不是令人费解,系统是不是必须转换到它的分支上来完成工作。

Jamie: 场景不就是用例的别名吗?

Vinod: 不是的,在这里场景具有不同的意义。

Doug: 你们谈论的是质量场景或者变更场景,是吗?

Vinod: 是的,我们要做的就是回到利益相关者那里,问问他们 SafeHome 在未来三年可能会有什么变更,如新版本、特征等这类变更。我们创建了一套变更场景,也开发了一套质量场景,这些场景定义了软件体系结构中要看到的属性。

Jamie: 我们把它运用到体系结构中。

Vinod: 是的,能更好地处理用例和场景的体系结构就是我们的最终选择。

12.7.1 体系结构描述语言

体系结构描述语言 (Architectural Description Language, ADL) 提供了一种描述软件体系结构的语义和语法。Hofmann 和他的同事 [Hof01] 建议 ADL 应该使设计者具有分解体系结构构件、将单独构件组合成大的体系结构块以及描述构件之间接口(连接机制)的能力。一旦进行了描述,就建立了基于语言技术的体系结构设计,当设计演化时,则更可能建立起有

① 软件体系结构分析方法 (SAAM) 是 ATAM 的替代方案,值得对体系结构分析感兴趣的读者进行研究。可以从 www.sei.cum.edu/publications/articles/saam-metho-propert-sas.html 网站上下载关于 SAAM 的论文。

效的体系结构评估方法。

软件工具 体系结构描述语言

下面这些重要的 ADL 总结来自 Rickard Land[Lan02], 已得到作者的允许。需要说明的是, 前 5 个 ADL 是为了研究目的而开发的, 它们不是商业产品。

- xArch (<http://www.isr.uci.edu/projects/xarchuci/>) 是一种标准的、可扩展的基于 XML 的软件体系结构表示方法。
- UniCon ([www.cs.cmu.edu/~ UniCon](http://www.cs.cmu.edu/~UniCon/)) 是“一种体系结构描述语言, 旨在帮助设计者用他们发现的很有用的抽象形式定义软件体系结构。”
- Wright ([www.cs.cmu.edu/~ able/wright/](http://www.cs.cmu.edu/~able/wright/)) 是一种包含如下元素的形式化语言: 带有端口的构件, 带有角色的连

接件, 以及将角色连接到端口上的粘合剂。可以使用谓词语言规范体系结构风格, 从而允许静态检查, 以确定体系结构的一致性和完整性。

- Acme ([www.cs.cmu.edu/~ acme/](http://www.cs.cmu.edu/~acme/)) 是第二代 ADL, 换句话说, 它的目的是确定一种最少共同点 ADL。
- UML (www.uml.org/) 包括很多体系结构描述所需要的部分——过程、结点、视图等。对于非正式的描述, UML 是相当适用的, 因为它有广泛的理解标准。然而, UML 没有足够的能力进行体系结构的充分描述。

12.7.2 体系结构评审

体系结构评审是一种特定的技术性评审, 它提供了一种评估方法, 该方法可以评估软件体系结构满足系统质量需求(如可扩展性或性能)的能力以及识别任何潜在风险的能力。体系结构评审可以尽早检测到设计问题, 具有降低项目成本的潜能。

与需求评审会涉及所有利益相关者的代表不同, 体系结构评审往往只涉及软件工程团队成员, 并辅以独立的专家。业界最常用的体系结构评审技术有: 基于经验的推理^①、原型评估、情境评审(第 8 章)以及检查单的使用^②。许多体系结构的评审在项目生命周期的早期就开始了, 当在基于构件的设计(第 13 章)中需要新增构件或程序包时也应进行体系结构评审。软件工程师们在进行体系结构评审时, 有时会发现体系结构的工作成果中有所缺失或不足, 这样会使得评审难以完成[Bab09]。

12.8 经验学习

基于软件的系统是由具有各种各样不同需求和观点的人建立起来的。因此, 软件体系结构设计师应该在软件团队成员(及其他利益相关者)间凝聚共识, 以便为最终的软件产品达成体系结构愿景[Wri11]。

体系结构设计师在创建体系结构时往往关注系统的非功能性需求的长期性影响。高级管理人员在业务战略和战术目标的背景中评估体系结构。

网络资源 软件体系结构设计的经验学习可从 http://www.sei.cmu.edu/library/abstracts/news-at-sei/01feature_200707.cfm 获得相关信息。

① 基于经验的推理是将新的体系结构与过往已使用的类似的体系结构进行对比。

② 典型的检查单可在此链接里找到: <http://www.opengroup.org/architecture/togaf7-doc/arch/p4/comp/clists/syseng.htm>。

项目经理经常受交付日期和预算等短期因素所驱使。软件工程师则常常关注他们自己的技术兴趣和所交付的功能。以上每个群体（以及其他相关人员）应致力于达成共识，以使所选定的软件体系结构明显优于其他可选的方案。

Wright [Wri11] 建议了几种决策分析和解决方案（Decision Analysis and Resolution, DAR）的方法，有助于消除分歧和促成协作。这些方法可以帮助增强团队成员的积极参与度，并提高他们认可最终决策的可能性。DAR 方法帮助团队成员以客观的方式来考虑几种可行的体系结构。三个代表性 DAR 方法如下：

- **原因链法**。一种根源分析^①的技术，团队定义好一种体系结构的目标或效果，然后阐述导致目标实现的相关动作。
- **石川鱼骨法**^②。一种图示技术，列出为达成既定的体系结构目标所需的各种可能的操作或原因。
- **思维导图或蜘蛛图**^③。这种图示方法用来表示围绕着一个中心关键词、约束或需求的词汇、概念、任务或软件工程产品。

网络资源 <http://www.infoq.com/articles/ieee-pattern-based-architecture-reviews> 有基于模式的体系结构评审的讨论。

12.9 基于模式的体系结构评审

正式技术评审可以应用于软件体系结构，从而为管理系统质量属性、发现错误以及避免不必要的返工提供一种手段。然而，现实情况中短暂的开发周期、紧迫的交付日期、反复变更的需求以及小规模的开发团队往往是常态。针对这些情况，一种轻量的、基于模式的体系结构评审（Pattern-Based Architecture Review, PBAR）流程可能是最佳的选择。

PBAR 是一种用来平衡体系结构模式^④与软件质量属性之间关系的评估方法。PBAR 是涉及所有开发者和其他有兴趣的利益相关者的面对面的审计会议。一位来自外部的体系结构、架构模式、质量属性以及应用领域的专业评审员也应该参加。系统体系结构设计师是首选的主持人。

应该在第一次工作原型或可运行的系统骨架^⑤完成后计划一次 PBAR。PBAR 包含以下的迭代步骤 [Har11]：

1. 遍历相关的用例（第 8 章），以确定并讨论系统最重要的质量属性。
2. 结合需求讨论系统体系结构图。
3. 协助评审人员识别所使用的体系结构模式，并将系统结构与模式结构相匹配。
4. 使用现有文档和过往用例，检查体系结构和质量属性，以确定每一种模式对系统质量

① 更多的信息可在此处获取：<http://www.thinkreliability.com/Root-Cause-Analysis-CM-Basics.aspx>。

② 更多的信息可在此处获取：<http://asq.org/learn-about-quality/cause-analysis/tools/overview/fishbone.html>。

③ 更多的信息可在此处获取：<http://mindmappingsoftwareblog.com/5-best-mind-mapping-programs-for-brainstorming/>。

④ 体系结构模式是针对一个体系结构设计问题以及一组特定的条件和约束的通用方案，模式在第 16 章中详细讨论。

⑤ 可运行的系统骨架包含一个支持业务案例中最高优先级的功能需求以及最有挑战性的质量属性的基线体系结构。

属性的影响。

5. 识别并讨论由设计中使用的体系结构模式所引起的问题。

6. 针对会议上出现的问题作一个简短的汇总，并对可运行的系统骨架进行相应的修正。

PBAR 非常适用于小规模敏捷团队，只需要相对较少的额外项目时间和精力。只需要很短的准备及评审时间，PBAR 便能够适应不断变更的需求和较短的建设周期，同时，也有助于团队理解系统体系结构。

12.10 体系结构一致性检查

随着软件的进程由设计进入到构建阶段，软件工程师们必须努力确保实施和演变中的系统与规划的体系结构是一致的。许多情形（如需求冲突、技术困难、交付期限的压力）会造成软件偏离原定的体系结构。如果没有定期对体系结构进行一致性检查，无法控制的偏差便会导致体系结构逐渐被侵蚀并影响系统质量 [Pas10]。

静态体系结构一致性分析（Static Architecture-Conformance Analysis, SACA）可以评估已完成的软件系统是否与它的体系结构模型相符合。系统体系结构建模的形式化方法（如 UML）表现了系统构件的静态组织以及构件间的交互。项目经理常用体系结构模型来分配工作任务和评估实施进程。

网络资源 <http://www.cin.ufpe.br/~fcf3/Arquitetura%20de%20Software/arquitetura/getPDF3.pdf> 有体系结构一致性检查的概览。

软件工具 | 体系结构一致性工具

- Lattix 依赖管理器 (<http://www.lattix.com/>)。这个工具包括一种简单语言，以声明实施中必须遵从的设计规则，检测设计规则中的违例，并按依存结构矩阵可视化地表现它们。
- 源代码查询语言 (<http://www.semmle.com/>)。这个工具可用来自动化软件开发任务，如定义并检查体系结构的约束，使用类 Prolog 语言定义面向对象系统的

继承层次结构的递归查询。

- Reflexion 模型 (http://www.iese.fraunhofer.de/en/competencies/architecture/tools_architecture.html#contentPar_textblockwithpics)。SAVE 工具允许软件工程师建立高级模型以抓取系统体系结构，然后定义这个模型与源代码之间的关系。然后 SAVE 将识别模型和代码之间的缺陷或错误。

12.11 敏捷性与体系结构

在一些敏捷开发的支持者眼里，体系结构设计等同于“大设计前期”，他们认为，这会产生不必要的文档和实现不必要的功能。然而，大多数敏捷开发者确实同意在系统复杂的情况下（即产品有大量的需求、许多利益相关者或广泛的地理分布），专注于软件体系结构是重要的 [Fal10]。基于此，有必要在敏捷过程模型中整合新的体系结构设计。

为了作早期的体系结构决策和避免返工需求，以及避免在选择了错误的体系结构时遭遇质量问题，敏捷开发者应基于一个新近发生的用户故事

网络资源 有关体系结构在敏捷软件过程中的角色可在 <http://msdn.microsoft.com/enus/architecture/ff476940.aspx> 中找到。

集来预料体系结构元素^①和结构（第5章）。敏捷团队可以通过建立一个体系结构原型（如一个可运行的系统骨架）并开发清晰的体系结构工作产品，与必要的利益相关者进行沟通，以满足体系结构设计需要。

敏捷开发促使软件体系结构设计师与业务和技术团队在迭代过程中反复地紧密合作，以指导良好的体系结构设计方向。Madison [Mad10] 建议使用包含 Scrum、XP 和顺序项目管理要素^②的混合框架。在这个框架中，前期规划设定了体系结构方向，但迅速进入故事情节串联 [Bro10b]。

在故事情节串联中，体系结构设计师为项目分享体系结构的用户故事，并在规划好“冲刺”（工作单元）后，与产品所有者一起对体系结构故事连同业务用户故事排列优先次序。在“冲刺”期间，体系结构设计师与团队一起工作，以确保演变的软件持续表现出较高的结构质量。如果质量水准高，团队可以独立完成剩余的开发；否则，体系结构设计师在“冲刺”期间加入团队。工作单元结束后，体系结构设计师将会在团队向利益相关者正式展示单元工作成果之前复审工作原型的质量。运作良好的敏捷项目要求每次“冲刺”迭代都应交付工作产品（包括体系结构文档）。对每一次“冲刺”浮现出的工作产品和代码进行复审是一种有用的体系结构评审方式。

职责驱动的体系结构（Responsibility-Driven Architecture, RDA）是一个专注于体系结构决策制定的过程。它指出体系结构决策应当在何时制定、如何制定以及由项目团队中的谁来制定。这种方法强调体系结构设计师是团队的服务型领导，而不是独裁的决策制定者，这与敏捷哲学是一致的。体系结构设计师作为主持者，关注并促进开发团队与团队之外的利益相关者（如业务、安全、基础架构等各类人员）之间的合作。

敏捷团队坚持在新的需求出现时自由地做出变更。体系结构设计师想要确保的是体系结构的重要部分已经经过了斟酌，并且开发者也征求了利益相关者的意见。两方的意见可以通过运用一种称为进展的签署的做法来得到满足，即在每一次新的原型完成后，相关的产品应该记录在文档中并获得批准 [Bla10]。

运用与敏捷思想一致的过程可给监管方和审计部门提供可验证的签署，而且不会妨碍敏捷团队做出所需的决策。在项目结束时，团队会有一整套工作产品，体系结构也会在演变过程中获得质量评审。

习题与思考题

- 12.1 用一个房屋或建筑物的结构作比喻，与软件体系结构作对照分析。经典建筑与软件体系结构的原则有什么相似之处？又有何区别？
- 12.2 举出 2～3 个例子，说明 12.3.1 节中提到的每一种体系结构风格的应用。
- 12.3 12.3.1 节中提到的一些体系结构风格具有层次性，而另外一些则没有。列出每种类型。没有层次的体系结构风格如何实现？
- 12.4 在软件体系结构讨论中，经常会遇到体系结构风格、体系结构模式及框架（本书中没有讨论）等术语。研究并描述这些术语之间的不同。

① 卓越的关于体系结构敏捷性的讨论可在 [Bro10a] 中找到。

② Scrum 和 XP 是敏捷过程模型，在第 5 章讨论。

- 12.5 选择一个你熟悉的应用软件, 回答 12.3.3 节中对于控制与数据提出的每一个问题。
- 12.6 研究 ATAM (使用 [Kaz98]) 并对 12.7.1 节提出的 6 个步骤进行详细讨论。
- 12.7 如果还没有完成问题 8.5, 请先完成它。使用本章描述的设计方法开发 PHTRS 的软件体系结构。
- 12.8 使用 12.1.4 节中的体系结构决策模板为问题 12.7 中的一个 PHTRS 体系结构决策撰写文档。
- 12.9 选取一个你熟悉的移动 App, 使用 12.4 节中的体系结构要素 (经济性、易见性、隔离性、对称性、应急性) 来对其进行评估。
- 12.10 列出问题 12.7 中你完成的 PHTRS 体系结构的优点及不足之处。
- 12.11 为问题 12.7 中你完成的 PHTRS 体系结构创建一个依赖结构矩阵^①。
- 12.12 从第 5 章中选取一种敏捷过程模型, 并标识出它所包含的体系结构设计活动。

扩展阅读与信息资源

在过去的 10 年中, 已经有很多关于软件体系结构的文献。Varma (《Software Architecture: A Case Based Approach》, Pearson, 2013) 通过一系列的案例研究来展现体系结构。Bass 和他的同事 (《Software Architecture in Practice》, 3rd ed., Addison-Wesley, 2012), Gorton (《Essential Software Architecture》, 2nd ed., Springer, 2011)、Rozanski 和 Woods (《Software Systems Architecture》, 2nd ed., Addison-Wesley, 2011)、Eeles 和 Cripps (《The Process of Software Architecting》, Addison-Wesley, 2009)、Taylor 和他的同事 (《Software Architecture》, Wiley, 2009)、Reekie 和 McAdam (《A Software Architecture Primer》, 2nd ed., Angophora Press, 2006) 以及 Albin (《The Art of Software Architecture》, Wiley, 2003) 的书中, 对智能领域挑战性的话题给出了有意义的介绍。

Buschman 和他的同事 (《Pattern-Oriented Software Architecture》, Wiley, 2007) 以及 Kuchana (《Software Architecture Design Patterns in Java》, Auerbach, 2004) 讨论了体系结构设计的面向模式的方面。Knoernschilf (《Java Application Architecture: Modularity Patterns with Examples Using OSGi》, Prentice Hall, 2012)、Rozanski 和 Woods (《Software Systems Architecture》, 2nd ed., Addison-Wesley, 2011)、Henderikson (《12 Essential Skills for Software Architects》, Addison-Wesley, 2011)、Clements 和他的同事 (《Documenting Software Architecture: View and Beyond》, 2nd ed., Addison-Wesley, 2010)、Microsoft (《Microsoft Application Guide》, Microsoft Press, 2nd ed., 2009)、Fowler (《Patterns of Enterprise Application Architecture》, Addison-Wesley, 2003)、Bosch[Bos00] 以及 Hofmeister 和他的同事 [Hof00] 提供了软件体系结构的更深入的方法。

Hennesey 和 Patterson (《Computer Architecture》, 5th ed., Morgan-Kaufmann, 2011) 对软件体系结构的设计问题作了清楚的定量描述。Clements 和他的同事 (《Evaluating Software Architecture》, Addison-Wesley, 2002) 对软件体系结构候选方案评估以及给定问题域的最优软件体系结构选取的相关问题进行了研究。

有关体系结构具体实现的书目陈述了特定开发环境和技术中的体系结构设计。Erl (《SOA Design Patterns》, Prentice-Hall, 2009) 以及 Marks 和 Bell (《Service-Oriented Architecture》, Wiley, 2006) 讨论了将业务及计算资源与客户定义需求联系起来的设计方法。Bambilla 等人 (《Model-Driven Software Engineering in Practice》, Morgan Claypool, 2012) 以及 Stahl 和他的同事 (《Model-Driven Software Development》, Wiley, 2006) 讨论了在具体领域建模方法环境中的体系结构。Radaideh 和 Al-ameed (《Architecture of Reliable Web Applications Software》, IGI Global, 2007) 研究了适用于 WebApp 的体

① 可以维基百科为起点获取更多关于 DSM 的资讯: http://en.wikipedia.org/wiki/Design_structure_matrix。

系结构。Esposito (《Architecting Mobile Solutions for the Enterprise》, Microsoft Press, 2012) 讨论了移动 App 体系结构。Clements 和 Northrop (《Software Product Lines: Practices and Patterns》, Addison-Wesley, 2001) 论述了支持软件产品线的体系结构。Shanley (《Protected Mode Software Architecture》, Addison-Wesley, 1996) 给出了设计基于 PC 的实时操作系统、多任务操作系统或者设备驱动程序的体系结构设计指导原则。

当前的软件体系结构研究每年都会记录在由 ACM 和其他计算机组织发起的软件体系结构国际研讨会会议录 (Proceedings of the International Workshop on Software Architecture) 和软件工程国际会议会议录 (Proceedings of the International Conference on Software Engineering) 中。

关于体系结构设计的大量的信息资源可以在网上获得。在 SEPA 网站 www.mhhe.com/pressman 上可以找到和体系结构设计相关的最新参考文献。

构件级设计

要点浏览

概念：完整的软件构件是在体系结构设计过程中定义的。但是没有在接近代码的抽象级上表示内部数据结构和每个构件的处理细节。构件级设计定义了数据结构、算法、接口特征和分配给每个软件构件的通信机制。

人员：软件工程师完成构件级设计。

重要性：必须能够在构造软件之前就确定该软件是否可以工作。为了保证设计的正确性，以及与早期设计表示（即数据、体系结构和接口设计）的一致性，构件级设计需要以一种可以评审设计细节的方式来表示软件。它提供了一种评估数据结构、接口和算法是否能够工作的方法。

步骤：数据、体系结构和接口的设计表示构成了构件级设计的基础。每个构件的

类定义或者处理说明都转化为一种详细设计，该设计采用图形或基于文本的形式来详细说明内部的数据结构、局部接口细节和处理逻辑。设计符号包括 UML 图和一些辅助表示，通过使用一系列结构化编程结构来说明过程的设计。通常的做法是采用现有可复用软件构件，而不是开发新的构件。

工作产品：每个构件的设计都以基于图形的表格或文本方式表示，这是构件级设计阶段产生的主要工作产品。

质量保证措施：采用设计评审机制。对设计执行检查以确定数据结构、接口、处理顺序和逻辑条件等是否都正确，并且给出早期设计中与构件相关的数据或控制变换。

体系结构设计第一次迭代完成之后，就应该开始构件级设计。在这个阶段，全部数据和软件的程序结构都已经建立起来。其目的是把设计模型转化为运行软件。但是现有设计模型的抽象层次相对较高，而可运行程序的抽象层次相对较低。这种转化具有挑战性，因为可能会在软件过程后期引入难以发现和改正的微小错误。Edsger Dijkstra（帮助我们理解软件设计技术的主要贡献者）在其著作 [Dij72] 中写道：

软件似乎不同于很多其他产品，对那些产品而言，一条规则是：更高的质量意味着更高的价格。那些想要真正可靠软件的人发现他们必须找到某种方法来避免开始时的大多数错误，结果，程序设计过程的成本更低……高效率的程序员……不应该将他们的时间浪费在调试上——在开始时就不应该引入错误。

尽管这段话是在很多年前说的，但现在仍然适用。当设计模型被转化为源代码时，必须遵循一系列设计原则，以保证不仅能够完成转化任务，而且不在开始时就引入错误。

关键概念

内聚性。
构件
适应性
分类
组合
合格性
基于构件的开发
内容设计
耦合
依赖倒置原则
设计重用
设计准则
领域工程

13.1 什么是构件

通常来讲,构件是计算机软件中的一个模块化的构造块。再正式一点,OMG 统一建模语言规范 [OMG03a] 是这样定义构件的:系统中模块化的、可部署的和可替换的部件,该部件封装了实现并对外提供一组接口。

正如第 12 章的讨论,构件存在于软件体系结构中,因而构件在完成所建系统的需求和目标的过程中起着重要作用。由于构件驻留于软件体系结构的内部,因此它们必须与其他的构件和存在于软件边界以外的实体(如其他系统、设备和人员)进行通信和合作。

对于术语构件的实际意义,软件工程师可能有不同的见解。在接下来的几节中,我们将了解关于“什么是构件以及在设计建模中如何使用构件”的三种重要观点。

关键概念

接口分离原则
Liskov 替换原则
面向对象观点
开闭原则
过程相关
传统构件
传统观点

引述 细节不仅是细节,它们构成设计。

Charles Eames

13.1.1 面向对象的观点

在面向对象软件工程环境中,构件包括一个协作类集合^①。构件中的每个类都得到详细阐述,包括所有属性和与其实现相关的操作。作为细节设计的一部分,必须定义所有与其他设计类相互通信协作的接口。为此,设计师需要从分析模型开始,详细描述分析类(对于构件而言该类与问题域相关)和基础类(对于构件而言该类为问题域提供了支持性服务)。

关键点 以面向对象的观点来看,构件是协作类的集合。

为了说明设计细化过程,考虑为一个高级影印中心构造软件。软件的目的是收集前台的客户需求,对印刷业务进行定价,然后把印刷任务交给自动生产设备。在需求工程中得到了一个名为 PrintJob 的分析类。分析过程中定义的属性和操作在图 13-1 的上方给出了注释。在体系结构设计中,PrintJob 被定义为软件体系结构的一个构件,用简化的 UML 符号表示的该构件显示在图 13-1 中部靠右的位置^②。需要注意的是,PrintJob 有两个接口:computeJob 和 initiateJob。computeJob 具有对任务进行定价的功能,initiateJob 能够把任务传给生产设备。这两个接口在图下方的左边给出(即所谓的棒棒糖式符号)。

构件级设计将由此开始。必须对 PrintJob 构件的细节进行细化,以提供指导实现的充分信息。通过不断补充作为构件 PrintJob 的类的全部属性和操作,来逐步细化最初的分析类。正如图 13-1 右下部分的描述,细化后的设计类 PrintJob 包含更多的属性信息和构件实现所需要的更广泛的操作描述。computeJob 和 initiateJob 接口隐含着与其他构件(图中没有显示出来)的通信和协作。例如,computePageCost() 操作(computeJob 接口的组成部分)可能与包含任务定价信息的 PricingTable 构件进行协作。checkPriority() 操作(initiateJob 接口的组成部分)可能与 JobQueue 构件进行协作,用来判断当前等待生产的任务类型和优先级。

建议 请回想一下,分析建模和设计建模都是迭代动作。细化原分析类可能需要额外的分析步骤,表示细化设计类的设计建模步骤紧随其后(构件的细节)。

对于体系结构设计组成部分的每个构件都要实施细化。细化一旦完成,要对每个属性、每个操作和每个接口进行更进一步的细化。对适合每个属性的数据结构

① 在某些情况下,构件可以包含一个简单的类。

② 不熟悉 UML 表示法的读者可以参考附录 1。

必须予以详细说明。另外还要说明实现与操作相关的处理逻辑的算法细节，在这一章的后半部分将要对此种过程设计活动进行讨论。最后是实现接口所需机制的设计。对于面向对象软件，还包含对实现系统内部对象间消息通信机制的描述。

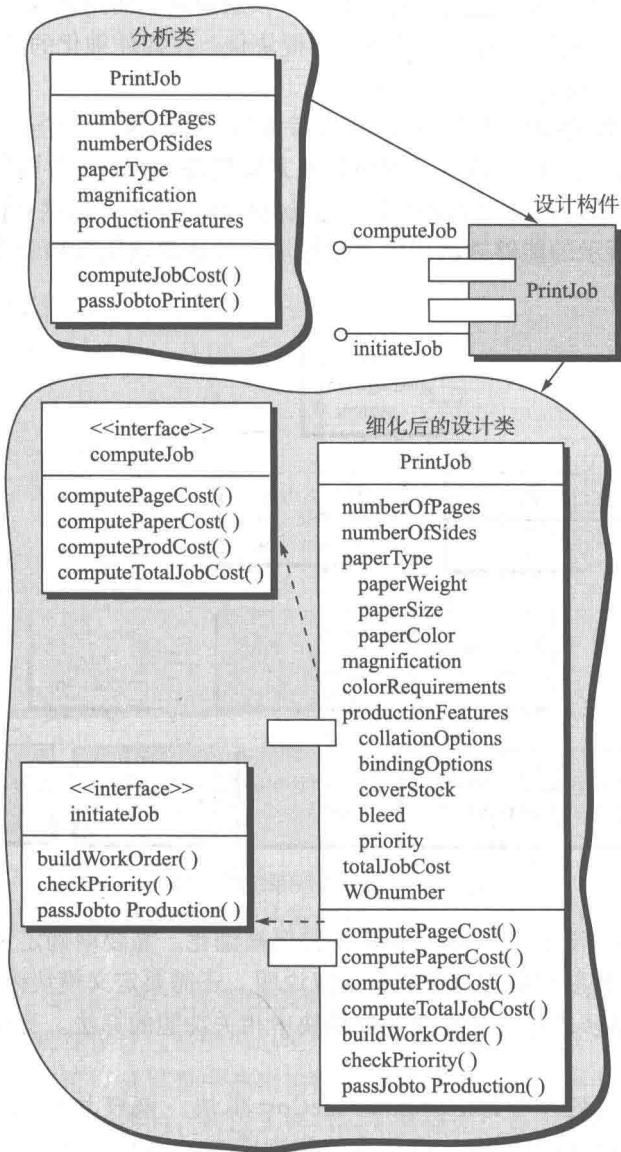


图 13-1 设计构件的细化

13.1.2 传统的观点

在传统软件工程环境中，一个构件就是程序的一个功能要素，程序由处理逻辑及实现处理逻辑所需的内部数据结构以及能够保证构件被调用和实现数据传递的接口构成。传统构件也称为模块，作为软件体系结构的一部分，它扮演如下三个重要角色之一：（1）控制构件，协调问题域中所有其他构件的调用；（2）问题域构件，完成客户需要的全部功能或部分功能；（3）基础设施构件，负责完成问题域中所需的支持处理的功能。

与面向对象的构件类似，传统的软件构件也来自于分析模型。不同的是在这种情况下，是以分析模型中的构件细化作为导出构件的基础。构件层次结构上的每个构件都被映射为某一层次上的模块（12.6 节）。一般来讲，控制构件（模块）位于层次结构（体系结构）顶层附近，而问题域构件则倾向位于层次结构的底层。为了获得有效的模块化，在构件细化的过程中采用了功能独立性的设计概念（第 11 章）。

为了说明传统构件的细化过程，我们再来考虑为一个高级影印中心构造的软件。一个分层的体系结构的导出如图 13-2 所示。图中每个方框都表示一个软件构件。带阴影的方框在功能上相当于 13.1.1 节讨论的为 PrintJob 类定义的操作。然而，在这种情况下，每个操作都被表示为如图 13-2 所示的能够被调用的单独模块。其他模块用来控制处理过程，也就是前面提到的控制构件。

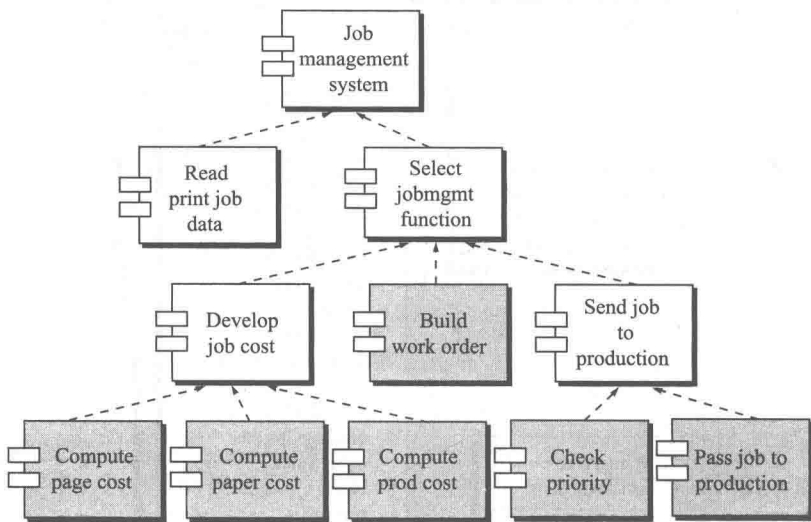


图 13-2 一个传统系统的结构图

在构件级设计中，图 13-2 中的每个模块都要被细化。需要明确定义模块的接口，即每个经过接口的数据或控制对象都需要明确加以说明。还需要定义模块内部使用的数据结构。采用第 11 章讨论的逐步求精方法设计完成模块中相关功能的算法。有时候需要用状态图表示模块行为。

为了说明这个过程，考虑 ComputePageCost 模块。该模块的目的在于根据用户提供的规格说明来计算每页的印刷成本。为了实现该功能需要以下数据：文档的页数、文档的印刷份数、单面或者双面印刷、颜色、纸张大小。这些数据通过该模块的接口传递给 ComputePageCost。ComputePageCost 根据任务量和复杂度，使用这些数据来决定一页的成本——这是一个通过接口将所有数据传递给模块的功能。每一页的成本与任务量成反比，与任务的复杂度成正比。

图 13-3 给出了使用改进的 UML 建模符号描述的构件级设计。其中 ComputePageCost 模块通过调用 getJobData 模块（它允许所有相关数据都传递给该构件）和数据库接口 accessCostDB（它能够使该模块访问存放所

引述 可工作的复杂系统都是由可工作的简单系统演化来的。

John Gall

建议 随着每个软件构件设计的逐步细化，设计焦点转向特定数据结构的设计和操作系统数据结构的过设计。但是，不要忘了体系结构，它必须容纳构件或服务于多数构件的全局数据结构。

有印刷成本的数据库)来访问数据。接着,对 ComputePageCost 模块进一步细化,给出算法和接口的细节描述(图 13-3)。其中,算法的细节可以由图中显示的伪代码或者 UML 活动图来表示。接口被表示为一组输入和输出的数据对象或者数据项的集合。设计细化的过程一直进行下去,直到能够提供指导构件构造的足够细节为止。

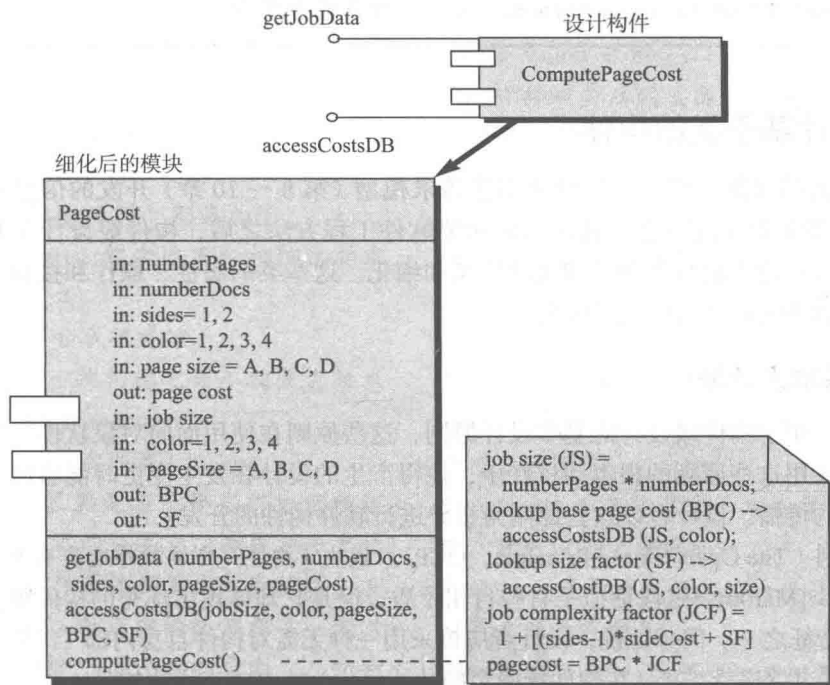


图 13-3 ComputePageCost 的构件级设计

13.1.3 过程相关的观点

13.1.1 节和 13.1.2 节提到的关于构件级设计的面向对象观点和传统观点,都假定从头开始设计构件。也就是说,设计者必须根据从需求模型中导出的规格说明创建新构件。当然,还存在另外一种方法。

在过去的 30 年间,软件工程已经开始强调使用已有构件或设计模式来构造系统的必要性。实际上,软件工程师在设计过程中可以使用已经过验证的设计或代码级构件目录。当软件体系结构设计完后,就可以从目录中选出构件或者设计模式,并用于组装体系结构。由于这些构件是根据复用思想来创建的,所以其接口的完整描述、要实现的功能和需要的通信与协作等对于设计者来说都是可以得到的。在 13.5 节将讨论基于构件的软件工程(Component-Based Software Engineering, CBSE)的某些重要方面。

信息栏 基于构件的标准和框架

基于构件的软件工程(CBSE)成功或者失败的重要因素之一就是基于构件标准(有时称为中间件)的可用性。中间件是一组基础构件的集合,这些基础构件使得问

题域构件与其他构件通过网络进行通信,或者在一个复杂的系统中通信。对于想用基于构件的软件工程方法进行开发的软件工程师来讲,他们可以使用如下的标准:

- OMG CORBA——<http://www.corba.org/>。
- Microsoft COM——<http://www.microsoft.com/com/default.mspx>。
- Microsoft .NET——<http://msdn.microsoft.com/en-us/netframework/default.aspx>。
- Sun JavaBeans——<http://www.oracle.com/technetwork/java/javaee/ejb/index.html>。

上面提到的网站上提供了大量的教材、白皮书、工具和关于这些重要中间件标准的大量资源。

13.2 设计基于类的构件

正如前面提到的，构件级设计利用了需求模型（第 8～10 章）开发的信息和体系结构模型（第 12 章）表示的信息。选择面向对象软件工程方法之后，构件级设计主要关注需求模型中问题域特定类的细化和基础类的定义和细化。这些类的属性、操作和接口的详细描述是开始构造活动之前所需的设计细节。

13.2.1 基本设计原则

有四种适用于构件级设计的基本设计原则，这些原则在使用面向对象软件工程方法时被广泛采用。使用这些原则的根本动机在于，使得产生的设计在发生变更时能够适应变更并且减少副作用的传播。设计者以这些原则为指导进行软件构件的开发。

开闭原则（The Open-Closed Principle, OCP）。模块（构件）应该对外延具有开放性，对修改具有封闭性 [Mar00]。这段话似乎有些自相矛盾，但是它却体现出优秀的构件级设计应该具有的最重要特征之一。简单地说，设计者应该采用一种无需对构件自身内部（代码或者内部逻辑）做修改就可以进行扩展（在构件所确定的功能域内）的方式来说明构件。为了达到这个目的，设计者需要进行抽象，在那些可能需要扩展的功能与设计类本身之间起到缓冲区的作用。

例如，假设 SafeHome 的安全功能使用了对各种类型安全传感器进行状态检查的 Detector 类。随着时间的推移，安全传感器的类型和数量将会不断增长。如果内部处理逻辑采用一系列 if-then-else 的结构来实现，其中每个这样的结构都负责一个不同的传感器类型，那么对于新增加的传感器类型，就需要增加额外的内部处理逻辑（依然是另外的 if-then-else 结构），而这显然违背 OCP 原则。

图 13-4 中给出了一种遵循 OCP 原则实现 Detector 类的方法。对于各种不同的传感器，Sensor 接口都向 Detector 构件呈现一致的视图。如果要添加新类型的传感器，那么对 Detector 类（构件）无需进行任何改变。这个设计遵守了 OCP 原则。

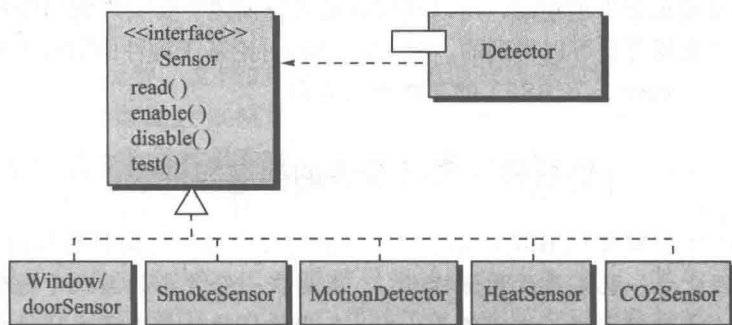


图 13-4 遵循 OCP 原则

SafeHome | OCP 的应用

[场景] Vinod 的工作间。

[人物] Vinod 和 Shakira, SafeHome 软件工程团队成员。

[对话]

Vinod : 我刚刚接到 Doug (团队经理) 的一个电话, 他说市场营销人员想增加一个新的传感器。

Shakira (假笑): 哎呀, 别再加了。

Vinod : 是啊……你永远不会相信这些家伙都提出了什么。

Shakira : 确实令我很吃惊。

Vinod (大笑): 他们称之为小狗焦虑传感器 (doggie angst sensor)。

Shakira : 那是什么装置?

Vinod : 这个装置是为了那些想把宠物留在彼此相邻很近的公寓、门廊或房子里的人设计的。狗叫致使邻里生气和抱怨, 有了这种传感器, 如果狗的叫声超过一定时间 (比如一分钟), 传感器就会向主人的手机发送特殊的报警信号。

Shakira : 你在开玩笑吗?

Vinod : 不是, Doug 想知道在安全功能中加入这个功能需要多长时间。

Shakira (想了想): 不用多长时间……瞧 (她给 Vinod 看图 13-4), 我们分离出在 sensor 接口背后的实际的传感器类。只要我们有小狗传感器的规格说明, 那么把它加入其中就是一件简单的事情了。我们要做的就是为其创建一个合适的构件……哦, 类。根本不用改变 Detector 构件。

Vinod : 好的, 我将告诉 Doug 这不是什么大问题。

Shakira : 告诉 Doug, 直到下一个版本发布之前, 我们都要集中精力完成小狗焦虑传感器的事情。

Vinod : 这不是件坏事, 而如果他想让你做, 你可以马上实现吗?

Shakira : 是啊, 我们的接口设计使得我可以毫无困难地完成它。

Vinod (想了想): 你听说过开闭原则吗?

Shakira (耸了耸肩膀): 没有。

Vinod (微笑): 不成问题。

Liskov 替换原则 (Liskov Substitution Principle, LSP)。子类可以替换它们的基类 [Mar00]。最早提出该设计原则的 Barbara Liskov [Lis88] 建议, 将从基类导出的类传递给构件时, 使用基类的构件应该仍然能够正确完成其功能。LSP 原则要求源自基类的任何子类必须遵守基类与使用该基类的构件之间的隐含约定。在这里的讨论中, “约定”既是前置条件——构件使用基类前必须为真; 又是后置条件——构件使用基类后必须为真。当设计者创建了导出类, 则这些子类必须遵守前置条件和后置条件。

依赖倒置原则 (Dependency Inversion Principle, DIP)。依赖于抽象, 而非具体实现 [MarR00]。正如我们在 OCP 中讨论的那样, 抽象可以比较容易地对设计进行扩展, 又不会导致大量的混乱。构件依赖的其他具体构件 (不是依赖抽象类, 如接口) 越多, 扩展起来就越困难。

接口分离原则 (Interface Segregation Principle, ISP)。多个客户专用接口比一个通用接口要好 [Mar00]。多个客户构件使用一个服务器类提供的操作的实例有很多。ISP 原则建议设计者应该为每个主要的客户类型都设计一个特定的接口。只有那些与特定客户类型相关的操作才应该出现在该客户的接口说明中。如果多个客户要求相同的操作, 则这些操作应该在每个特定的接口中都加以说明。

建议 如果你省去设计并且破解出代码, 记住代码只是最终的“具体实现”, 你违背了 DIP 原则。

例如, 假设 FloorPlan 类用在 SafeHome 的安全和监视功能中 (第 9 章)。对于安全功能,

FloorPlan 只在配置活动中使用,并且使用 placeDevice()、showDevice()、groupDevice()、removeDevice() 等操作实现在建筑平面图中放置、显示、分组和删除传感器。SafeHome 监视功能除了需要这四个有关安全的操作之外,还需要特殊的操作 showFOV()、showDeviceID() 来管理摄像机。因此,ISP 建议为来自 SafeHome 功能的两个客户端构件定义特殊的接口。安全接口应该只包括 placeDevice()、showDevice()、groupDevice()、removeDevice() 四种操作。监视接口应该包括 placeDevice()、showDevice()、groupDevice()、removeDevice()、showFOV()、showDeviceID() 六种操作。

关键点 设计可复用的构件不仅需要优秀的技术设计,而且需要高效的配置控制机制(第21章)。

尽管构件级设计原则提供了有益的指导,但构件自身不能够独立存在。在很多情况下,单独的构件或者类被组织到子系统或包中。于是我们很自然地就会问这个包会有怎样的活动。在设计过程中如何正确组织这些构件? Martin 在 [Mar00] 中给出了在构件级设计中可以应用的另外一些打包原则,这些原则如下。

发布复用等价性原则 (Release Reuse Equivalency Principle, REP)。复用的粒度就是发布的粒度 [Mar00]。当设计类或构件用以复用时,在可复用实体的开发者和使用者之间就建立了一种隐含的约定关系。开发者承诺建立一个发布控制系统,用来支持和维护实体的各种老版本,同时用户逐渐地将其升级到最新版本。明智的方法是将可复用的类分组打包成能够管理和控制的包并作为一个更新的版本,而不是对每个类分别进行升级。

共同封装原则 (Common Closure Principle, CCP)。一同变更的类应该合在一起 [Mar00]。类应该根据其内聚性进行打包。也就是说,当类被打包成设计的一部分时,它们应该处理相同的功能或者行为域。当某个域的一些特征必须变更时,只有相应包中的类才有可能需要修改。这样可以进行更加有效的变更控制和发布管理。

共同复用原则 (Common Reuse Principle, CRP)。不能一起复用的类不能被分到一组 [Mar00]。当包中的一个或者多个类变更时,包的发布版本号也会发生变更。所有那些依赖于已经发生变更的包的类或者包,都必须升级到最新版本,并且都需要进行测试以保证新发布的版本能够无故障运转。如果类没有根据内聚性进行分组,那么这个包中与其他类无关联的类有可能会发生变更,而这往往会导致进行没有必要的集成和测试。因此,只有那些一起被复用的类才应该包含在一个包中。

13.2.2 构件级设计指导方针

除了 13.2.1 节中讨论的原则之外,在构件级设计的进程中还可以使用一系列实用的设计指导方针。这些指导方针可以应用于构件、构件的接口,以及对于最终设计有着重要影响的依赖和继承特征等方面。Ambler [Amb02b] 给出了如下的指导方针。

构件。对那些已经被确定为体系结构模型一部分的构件应该建立命名约定,并对其做进一步的精细化处理,使其成为构件级模型的一部分。体系结构构件的名字来源于问题域,并且对于考虑体系结构模型的所有利益相关者来说都是有意义的。例如,无论技术背景如何, FloorPlan 这个类的名称对于任何读到它的人来说都是有意义的。另一方面,基础构件或者细化后的构件级类应该以能够反映其实现意义的名称来命名。例如,对一个作为 FloorPlan 实现一部分的链表进

提问 命名构件时需要考虑些什么?

行管理时, 操作 `manageList()` 是一个合适的名称, 因为即使是非技术人员也不会误解。^①

在详细设计层面使用构造型帮助识别构件的特性也很有价值。例如, `<<infrastructure>>` 可以用来标识基础设施构件; `<<database>>` 可以用来标识服务于一个或多个设计类或者整个系统的数据库; `<<table>>` 可以用来标识数据库中的表。

接口。接口提供关于通信和协作的重要信息(也可以帮助我们实现 OCP 原则)。然而, 接口表示的随意性会使构件图趋于复杂化。Ambler[Amb02c] 建议: (1) 当构件图变得复杂时, 在较正式的 UML 框和虚箭头记号方法中使用接口的棒棒糖式记号^②; (2) 为了保持一致, 接口都放在构件框的左边; (3) 即使其他的接口也适用, 也只表示出那些与构件相关的接口。这些建议意在简化 UML 构件图, 使其易于查看。

依赖与继承。为了提高可读性, 依赖关系自左向右, 继承关系自底(导出类)向上(基类)。另外, 构件之间的依赖关系通过接口来表示, 而不是采用“构件到构件”的方法来表示。遵照 OCP 的思想, 这种方法使得系统更易于维护。

13.2.3 内聚性

在第 11 章中, 我们将内聚性描述为构件的“专一性”。在面向对象系统进行构件级设计时, 内聚性意味着构件或者类只封装那些相互关联密切, 以及与构件或类自身有密切关系的属性和操作。Lethbridge 和 Laganière[Let01] 定义了许多不同类型的内聚性(按照内聚性的级别排序^③)。

功能内聚。主要通过操作来体现, 当一个模块完成一组且只有一组操作并返回结果时, 就称此模块是功能内聚的。

分层内聚。由包、构件和类来体现。高层能够访问低层的服务, 但低层不能访问高层的服务。例如, 如果警报响起, SafeHome 的安全功能需要打出一个电话。可以定义如图 13-5 所示的一组分层包, 带阴影的包中包含基础设施构件。访问都是从 Control panel 包向下进行的。

通信内聚。访问相同数据的所有操作被定义在一个类中。一般说来, 这些类只着眼于数据的查询、访问和存储。

那些体现出功能、层和通信等内聚性的类和构件, 相对来说易于实现、测试和维护。设计者应该尽可能获得这些级别的内聚性。然而, 需要强调的是, 实际的设计和实现问题有时会迫使设计者选择低级别的内聚性。

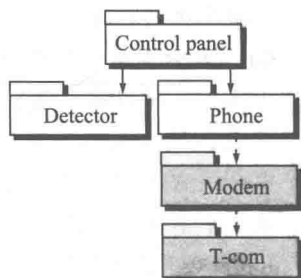


图 13-5 分层内聚

建议 尽管理解各种级别的内聚性很有益, 但是更为重要的是在设计构件时对内聚性有全面的理解。尽可能保持高内聚性。

SafeHome 内聚性的应用

[场景] Jamie 的工作间。

[对话]

[人物] Jamie 和 Ed, SafeHome 软件工程团队成员, 正在实现监视功能。

Ed: 我已经完成了 camera (摄像机) 构件的初步设计。

① 由销售部和顾客部(非技术类型)人员检查详细的设计信息是靠不住的。

② 这里指的是类似图 13-1 中的接口表示。——译者注

③ 一般来说, 内聚性级别越高, 构件实现、测试和维护起来就越容易。

Jamie: 希望快速评审一下吗?

Ed: 我想……但实际上,你最好输入一些信息。(Jamie 示意他继续。)

Ed: 我们起初为 camera 构件定义了 5 种操作。看……

determineType() 给出用户摄像机的类型。

translateLocation() 允许用户删除设计图上的摄像机。

displayID() 得到摄像机 ID,并将其显示在摄像机图标附近。

displayView() 以图形化方式给出用户摄像机的视角范围。

displayZoom() 以图形化方式给出用户摄像机的放大率。

Ed: 我分别进行了设计,并且它们非常易于操作。所以我认为,将所有的显示操作集成到一个称为 displayCamera() 的操作中——它可以显示 ID、视角和放大率等,是一个不错的主意。你不这样认为吗?

Jamie (扮了个鬼脸): 我不敢肯定。

Ed (皱眉): 为什么?所有这些小操作是很令人头疼的!

Jamie: 问题是当将它们集成到一起

后,我们将失去内聚性。你知道的,displayCamera() 操作不是专一的。

Ed (略有不快): 那又怎样?这样做使得我们最多只需不到 100 行的源代码。我想实现起来也比较简单。

Jamie: 如果销售人员决定更改我们显示视角域的方式怎么办?

Ed: 我马上跳到 displayCamera() 操作,并进行这种改变。

Jamie: 那么引起的副作用怎么办?

Ed: 什么意思?

Jamie: 也就是说你做了修改,但是不经意之间产生了 ID 的显示问题。

Ed: 我没有那么笨。

Jamie: 可能没有,但是如果两年以后某些支持人员必须做这种改变怎么办。他可能并不像你一样理解这个操作,谁知道呢,也许他很粗心。

Ed: 所以你反对?

Jamie: 你是设计师,这是你的决定,只要确信你理解了低内聚性的后果。

Ed (想了想): 可能我们要设计一个单独的显示操作。

Jamie: 好主意。

13.2.4 耦合性

在前面关于分析和设计的讨论中,我们知道通信和协作是面向对象系统中的基本要素。然而,这个重要(必要)特征存在一个黑暗面。随着通信和协作数量的增长(也就是说,随着类之间的联系程度越来越强),系统的复杂性也随之增长了。同时,随着系统复杂度的增长,软件实现、测试和维护的困难也随之增大。

耦合是类之间彼此联系程度的一种定性度量。随着类(构件)之间的相互依赖越来越多,类之间的耦合程度亦会增加。在构件级设计中,一个重要的目标就是尽可能保持低耦合。

有多种方法来表示类之间的耦合。Lethbridge 和 Laganière[Let01]定义了一组耦合分类。例如,内容耦合发生在当一个构件“暗中修改其他构件的内部数据”[Let01]时。这违反了基本设计概念当中的信息隐蔽原则。控制耦合发生在当操作 A 调用操作 B,并且向 B 传递了一个控制标记时。接着,控制标记将会指引 B 中的逻辑流程。这种耦合形式的主要问题在于,B 中的一个不相关变更往往能够导

建议 随着每个软件构件的细化,我们的关注点将转移到数据结构设计上,包括其相关的过程设计。但是,不要忘记体系结构,因为构件和服务于构件的全局数据结构都住在体系结构这所大房子里。

致 A 所传递控制标记的意义也必须发生变更。如果忽略这个问题,就会引起错误。外部耦合发生在当一个构件和基础设施构件(例如,操作系统功能、数据库容量、无线通信功能等)进行通信和协作时。尽管这种类型的耦合是必要的,但是在一个系统中应该尽量将这种耦合限制在少量的构件或者类范围内。

软件必须进行内部和外部的通信,因此,耦合是必然存在的。然而,在不可避免出现耦合的情况下,设计者应该尽力降低耦合性,并且要充分理解高耦合的后果。

SafeHome 耦合的应用

[场景] Shakira 的工作间。

[人物] Vinod 和 Shakira, SafeHome 软件工程团队成员,正在实现安全功能。

[对话]

Shakira: 我曾经有一个非常好的想法,但之后我又考虑了一下,觉得好像并没有那么好。最后我还是放弃了,不过我想最好和你讨论一下。

Vinod: 当然可以,是什么想法?

Shakira: 好的,每个传感器能够识别一种警报条件,对吗?

Vinod (微笑): 这就是我们称它为传感器的一个原因啊, Shakira。

Shakira (恼怒): 你讽刺我! 你应该好好学习一下处理人际关系的技巧。

Vinod: 你刚才说什么?

Shakira: 我指的是……为什么不为每个传感器都创建一个名为 makeCall() 的操作,该操作能够直接和 OutgoingCall (外呼)

构件协作,也就是通过 OutgoingCall 构件的接口实现协作?

Vinod (沉思着): 你的意思是让协作发生在 ControlPanel 之类的构件之外?

Shakira: 是的,但接着我又对自己说,这将意味着每个传感器对象都会与 OutgoingCall 构件相关联,而这意味着与外部世界的间接耦合……我想这样会使事情变得复杂。

Vinod: 我同意,在这种情况下,让传感器接口将信息传递给 ControlPanel,并且让其启动外呼,这是一个比较好的主意。此外,不同的传感器将导致不同的电话号码。在信息改变时,你并不希望传感器存储这些信息,因为如果发生变化……

Shakira: 感觉不太对。

Vinod: 耦合设计方法告诉我们是有点对。

Shakira: 无论如何……

13.3 实施构件级设计

在本章的前半部分,我们已经知道构件级设计本质上是精细化的。设计者必须将需求模型和架构模型中的信息转化为一种设计表示,这种表示提供了用来指导构件(编码和测试)活动的充分信息。应用于面向对象系统时,下面的步骤表示出构件级设计典型的任务集。

步骤 1: 标识出所有与问题域相对应的设计类。使用需求模型和架构模型,正如 13.1.1 节中所描述的那样,每个分析类和体系结构构件都要细化。

步骤 2: 确定所有与基础设施域相对应的设计类。在需求模型中并没有描述这些类,并且在体系结构设计中也经常忽略这些类,但是此时必须对它们进行描述。如前所述,这种类型的类和构件包括 GUI (图形用户界面) 构件(通常为可复用构件)、操作系统构件以及对象

引述 如果我有更多的时间,我将写一封更短的信。

Blaise Pascal

和数据管理构件等。

步骤 3：细化所有不需要作为复用构件的设计类。详细描述实现类细化所需要的所有接口、属性和操作。在实现这个任务时，必须考虑采用设计的启发式规则（如构件的内聚和耦合）。

步骤 3a：在类或构件协作时说明消息的细节。需求模型中用协作图来显示分析类之间的相互协作。在构件级设计过程中，某些情况下有必要通过对系统中对象间传递消息的结构进行说明来表现协作细节。尽管这是一个可选的设计活动，但是它可以作为接口规格说明的前提，这些接口显示了系统构件之间通信和协作的方式。

图 13-6 给出了前面提到的影印系统的一个简单协作图。ProductionJob、WorkOrder 和 JobQueue 这三个对象相互协作，为生产线准备印刷作业。图中的箭头表示对象间传递的消息。在需求建模时，消息说明如图 13-6 所示。然而，随着设计的进行，消息通过下列方式的扩展语法来细化 [Ben02]：

```
[guard condition] sequence expression (return value) :=
message name (argument list)
```

其中，[guard condition] 采用对象约束语言（Object Constraint Language, OCL）^①来书写，并且说明了在消息发出之前应该满足什么样的条件集合；sequence expression 是一个表明消息发送序号的整数（或其他样式的表明发送顺序的指示符，如 3.1.2）；(return value) 是由消息唤醒操作返回的信息名；message name 表示唤醒的操作，(argument list) 是传递给操作的属性列表。

步骤 3b：为每个构件确定适当的接口。在构件级设计中，一个 UML 接口是“一组外部可见的（即公共的）操作。接口不包括内部结构，没有属性，没有关联……”[Ben02]。更正式地讲，接口就是某个抽象类的等价物，该抽象类提供了设计类之间的可控连接。图 13-1 给出了接口细化的实例。实际上，为设计类定义的操作可以归结为一个或者多个抽象类。抽象类内的每个操作（接口）应该是内聚的，也就是说，它应该展示那些关注于一个有限功能或者子功能的处理。

参照图 13-1，initiateJob 接口由于没有展现出足够的内聚性而受到争议。实际上，它完成了三个不同的子功能：建立工作单，检查任务的优先级，并将任务传递给生产线。接口设计应该重构。一种方法就是重新检查设计类，并定义一个新类 WorkOrder，该类的作用就是处理与装配工作单相关的所有活动。操作 buildWorkOrder() 成为该类的一部分。类似地，我们可能需要定义包括操作 checkPriority() 在内的 JobQueue 类。ProductionJob 类包括给生产线传递生产任务的所有相关信息。initiateJob 接口将采用图 13-7 所示的形式。initiateJob 现在是内聚的，集中在一个功能上。与 ProductionJob、WorkOrder 和 JobQueue 相关的接口几乎都是专一的。

步骤 3c：细化属性，并且定义实现属性所需要的数据类型和数据结构。描述属性的数据类型和数据结构一般都需要在实现时所采用的程序设计语言中进行定义。UML 采用下面

建议 如果在非面向对象的环境下工作，那么前 3 步的重点在于提炼数据对象和处理功能（转换），这些功能被视为需求模型的一部分。

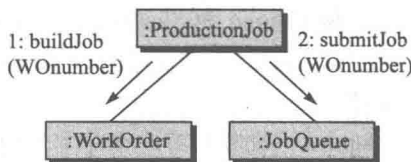


图 13-6 带消息的协作图

① OCL 在附录 1 中有简明的介绍。

的语法来定义属性的数据类型：

```
name:type-expression = initial-value{property string}
```

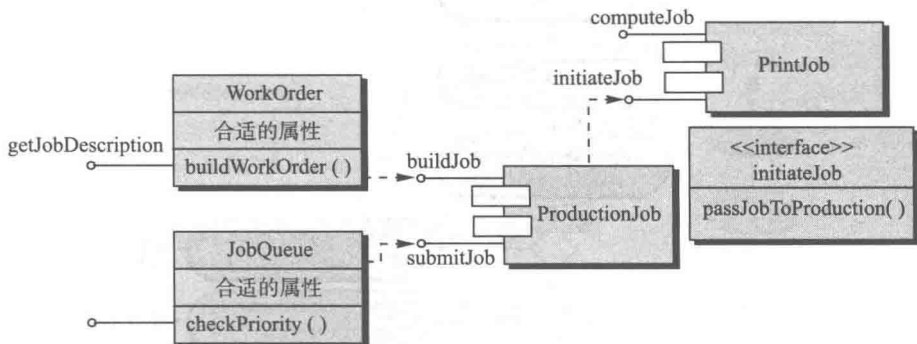


图 13-7 为 PrintJob 重构接口和类定义

其中，name 是属性名，type-expression 是数据类型，initial-value 是创建对象时属性的取值，property string 用于定义属性的特征或特性。

在构件级设计的第一轮迭代中，属性通常用名字来描述。再次参考图 13-1，PrintJob 的属性列表只列出了属性名。然而，随着设计的进一步细化，我们将使用 UML 的属性格式注释来定义每个属性。例如，以下列方式来定义 paperType-weight：

```
paperType-weight: string = "A" {contains 1 of 4 values-A, B, C, or D}
```

这里将 paperType-weight 定义为一个字符串变量，初始值为 A，可以取值为集合 {A, B, C, D} 中的一个。

如果某一属性在多个设计类中重复出现，并且其自身具有比较复杂的结构，那么最好是为这个属性创建一个单独的类。

步骤 3d：详细描述每个操作中的处理流。这可能需要由基于程序设计语言的伪代码或者由 UML 活动图来完成。每个软件构件都需要应用逐步求精概念（第 11 章）通过很多次迭代进行细化。

第一轮迭代中，将每个操作都定义为设计类的一部分。在任何情况下，操作应该确保具有高内聚性的特征；也就是说，一个操作应该完成单一的目标功能或者子功能。接下去的一轮迭代，只是完成对操作名的详细扩展。例如，图 13-1 中的操作 computePaperCost() 可以采用如下方式进行扩展：

```
computePaperCost(weight, size, color): numeric
```

这种方式说明 computePageCost() 要求属性 weight、size 和 color 作为输入，并返回一个数值（实际上为金额）作为输出。

如果实现 computePaperCost() 的算法简单而且易于理解，则没有必要开展进一步的设计细化。软件编码人员将会提供实现这些操作的必要细节。但是，如果算法比较复杂或者难于理解，此时则需要进一步的设计细化。图 13-8 给出了操作 computePaperCost() 的 UML 活动图。当活动图用于构件级设计的规格说明时，通常都在比源码更高的抽象级上表示。还有一种方法是在设计规格说明中使用伪代码。

建议 在细化构件设计时使用逐步求精的方法。经常提出这样的问题：“是否存在一种方法可以简化问题并仍能达到相同的结果？”

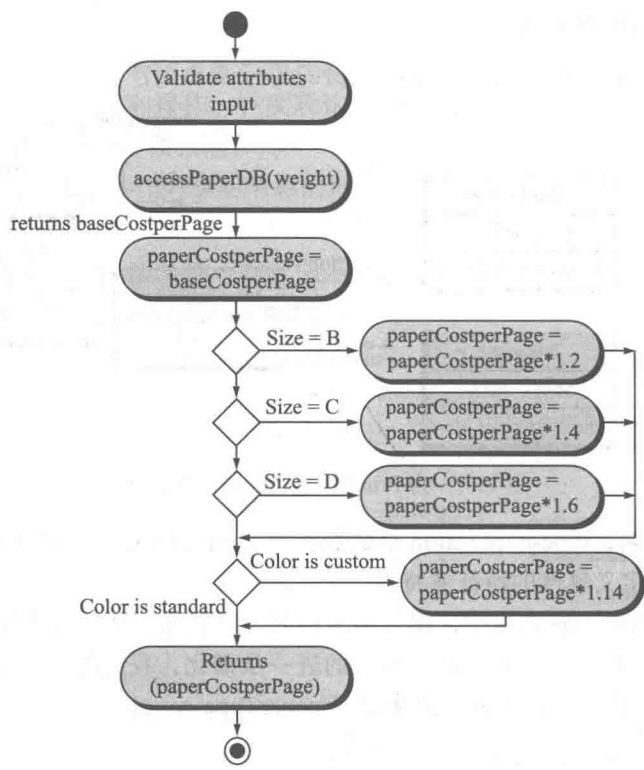


图 13-8 computePaperCost() 操作的 UML 活动图

步骤 4：说明持久数据源（数据库和文件）并确定管理数据源所需要的类。数据库和文件通常都凌驾于单独的构件设计描述之上。在多数情况下，这些持久数据存储最初都作为体系结构设计的一部分进行说明，然而，随着设计细化过程的不断深入，提供关于这些持久数据源的结构和组织等额外细节常常是有用的。

步骤 5：开发并且细化类或构件的行为表示。UML 状态图被用作需求模型的一部分，表示系统的外部可观察的行为和更多的分析类个体的局部行为。在构件级设计过程中，有些时候对设计类的行为进行建模是必要的。

对象（程序执行时的设计类实例）的动态行为受到外部事件和对象当前状态（行为方式）的影响。为了理解对象的动态行为，设计者必须检查设计类生命周期中所有相关的用例，这些用例提供的信息可以帮助设计者描述影响对象的事件，以及随着时间流逝和事件的发生对象所处的状态。图 13-9 描述了使用 UML 状态图 [Ben02] 表示的状态之间的转换（由事件驱动）。

从一种状态到另一种状态的转换（用圆角矩形来表示），都表示为如下形式的事件序列：

```
event-name (parameter-list) [guard-condition] / action expression
```

其中，event-name 表示事件；parameter-list 包含了与事件相关的数据；guard-condition 采用对象约束语言 OCL 书写，并描述了事件发生前必须满足的条件；action expression 定义了状态转换时发生的动作。

参照图 13-9，针对状态的进入和离开两种情形，每个状态都可以定义 entry/ 和 exit/ 两个动作。在大多数情况下，这些动作与正在建模的类的相关操作相对应。do/ 指示符提供了

一种机制，用来显示伴随此种状态的相关活动；而 include/ 指示符则提供了通过在状态定义中嵌入更多状态图细节的方式进行细化的手段。

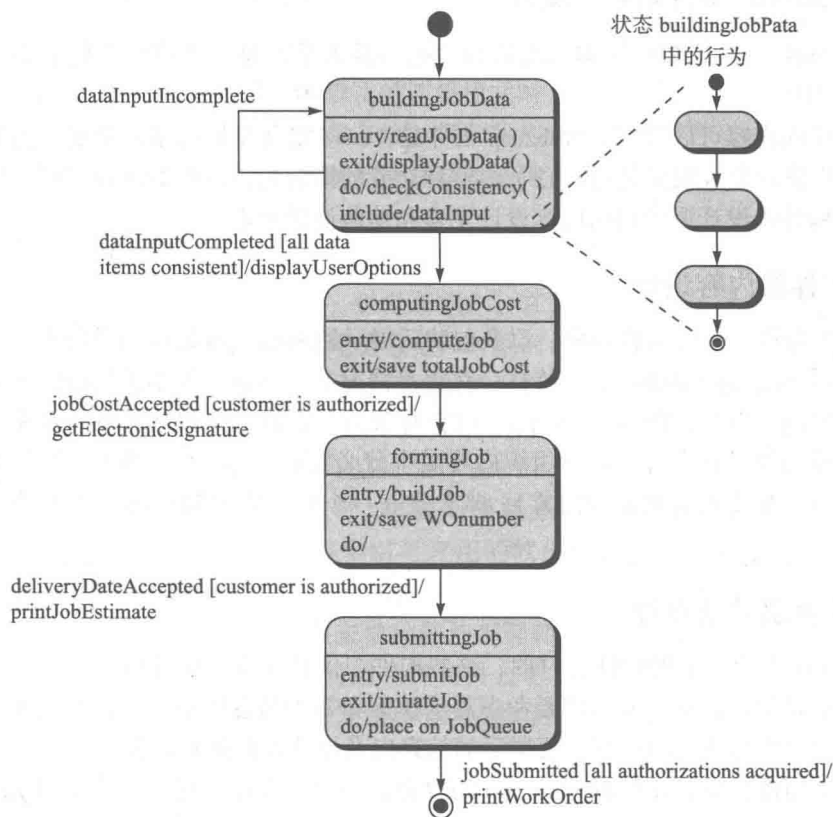


图 13-9 PrintJob 类的状态图

需要注意的重要一点是，行为模型经常包含一些在其他设计模型中不明显的信息。例如，通过仔细查看图 13-9 中的状态图可以知道，当得出印刷任务的成本和进度数据时，PrintJob 类的动态行为取决于用户对此是否认可。如果没有得到允许（警戒条件确保用户有权审核），印刷工作就不能提交，因为不可能到达 submittingJob 状态。

步骤 6：细化部署图以提供额外的实现细节。部署图（第 11 章）作为体系结构设计的一部分，采用描述符形式来表示。在这种表示形式中，主要的系统功能（经常表现为子系统）都表示在容纳这些功能的计算环境中。

在构件级设计过程中，应该对部署图进行细化，以表示主要构件包的位置。然而，一般在构件图中不单独表示构件，目的在于避免图的复杂性。某些情况下，部署图在这个时候被细化成实例形式。这意味着要对指定的硬件和使用的操作系统环境加以说明，而构件包在这个环境中的位置也需要确定。

步骤 7：考虑每个构件级设计表示，并且时刻考虑其他可选方案。纵观全书，我们始终强调设计是一个迭代过程。创建的第一个构件级模型总没有迭代多次之后得到的模型那么全面、一致或精确。在进行设计工作时，重构是十分必要的。

另外，设计者不能眼光狭隘。设计中经常存在其他的设计方案，在没有决定最终设计模型之前，最好的设计师会考虑所有（或大部分）的方案，运用第 11 章和本章介绍的设计原

则和概念开发其他的可选方案，并且仔细考虑和分析这些方案。

13.4 WebApp 的构件级设计

在基于 Web 的系统和应用中，内容和功能的界限常常是模糊的。因此有必要考虑什么是 WebApp 构件。

根据本章的内容可以知道，WebApp 构件是：（1）定义良好的聚合功能，为最终用户处理内容，或提供计算或数据处理；（2）内容和功能的聚合包，提供最终用户所需的功能。因此 WebApp 构件级设计通常包括内容设计元素和功能设计元素。

13.4.1 构件级内容设计

构件级内容设计关注内容对象，以及包装后展示给 WebApp 最终用户的方式。构件级内容设计应该适合创建的 WebApp 的特性。在很多情况下，内容对象不需要被组织成构件，它们可以分别实现。但是，随着 WebApp、内容对象以及它们之间相互关系的规模和复杂度的增长，在更好的参考和设计方法下组织内容是十分必要的^①。此外，如果内容显示出高度的动态性（如一个在线拍卖网站的内容），那么建立一个包含内容构件的、清晰的结构模型是非常重要的。

13.4.2 构件级功能设计

WebApp 是作为一系列构件交付的，这些构件与信息体系结构并行开发，以确保它们的一致性。最重要的是，在一开始就要考虑需求模型和初始信息体系结构，然后再进一步考查功能如何影响用户与系统的交互、要展示的信息以及要管理的用户任务。

在体系结构设计中，往往将 WebApp 的内容和功能结合在一起设计应用系统的功能体系结构。在这里，功能体系结构代表的是 WebApp 的功能域，并且描述了关键的功能构件和这些构件是如何进行交互的。

13.5 设计传统构件

传统软件构件^②的构件级设计基础在 20 世纪 60 年代早期已经形成，在 Edsger Dijkstra ([Dij65], [Dij76b]) 及他人的著作（如 [Boh66]）中又得到了进一步的完善。20 世纪 60 年代末，Dijkstra 等人提出，所有的程序都可以建立在一组限定好的逻辑构造上。这组逻辑构造强调“对功能域的支持”，其中每个逻辑结构都是可预测的，过程流从顶端进入，从底端退出，读者很容易理解。

这些结构包括顺序型、条件型和重复型。顺序型实现了任何算法规格说明中的核心处理步骤；条件型允许根据逻辑情况选择处理的方式；重复型提供了循环。这三种结构是结构化程序设计的基础，而结构化程序设计是一种重要的构件级设计技术。

结构化的构建使得软件的过程设计只采用少数可预知的逻辑结构。复

关键点 结构化程序设计是一种设计技术，该技术将程序逻辑流程限制为以下三种结构：顺序型，条件型和重复型。

① 内容构件可以在其他 WebApp 中复用。

② 传统的软件构件实现处理元素，这些处理元素涉及问题域中的功能或子功能，或者涉及基础设施域中的某种性能。通常将传统构件称为模块、程序或子程序，传统构件不像面向对象构件那样封装数据。

杂性度量表明,使用结构化的构造降低了程序复杂性,从而增加了可读性、可测试性和可维护性。使用有限数量的逻辑结构也符合心理学家所谓的人类成块的理解过程。要理解这一过程,可以考虑阅读英文的方式。读者不是阅读单个字母,而是辨认由单词或短语构成的模式或是字母块。结构化的构造就是一些逻辑块,读者可以用它来辨认模块的过程元素,而不必逐行阅读设计或是代码,当遇到了容易辨认的逻辑模式时,理解力就得到了提高。

无论是对于应用领域还是对于技术复杂度,任何程序都可以只用这三种结构化的构造来设计和实现。然而,需要注意的是,教条地使用这些结构在实践中会遇到困难。

13.6 基于构件的开发

在软件工程领域,复用既是老概念,也是新概念。在计算机发展的早期,程序员就已经复用概念、抽象和过程,但是早期的复用更像是一种临时的方法。今天,复杂的、高质量的计算机系统往往必须在短时间内开发完成,所以就需一种更系统的、更有组织性的复用方法来协助这样的快速开发。

基于构件的软件工程(Component-Based Software Engineering, CBSE)是一种强调使用可复用的软件构件来设计与构造计算机系统的过程。考虑到这种解释,很多问题出现了,仅仅将多组可复用的软件构件组合起来,就能够构造出一个复杂的系统吗?这种工作能够以一种高效和节省成本的方式完成吗?能否建立恰当的激励机制来鼓励软件工程师复用而不是重复开发?管理团队是否也愿意为构造可复用软件构件过程中的额外开销买单?能否以使用者易于访问的方式构造复用所必需的构件库?已有的构件可以被需要的人找到并使用吗?大家渐渐地发现了这些问题的答案,而且这些问题的答案都是“可以”。

引述 领域工程是发现系统间的共性,以识别可以应用于很多系统的构件,并识别最能充分利用那些构件的程序族。

Paul Clements

13.6.1 领域工程

领域工程的目的是识别、构造、分类和传播一组软件构件,这些构件在某一特定的应用领域中可以适用于现有和未来的软件^①。总体目标是为软件工程师建立一种机制来分享这些构件,从而在开发新系统或改造现有系统时可以共享这些构件——复用它们。领域工程包括三种主要活动:分析、构建和传播。

领域分析的总体方法通常在面向对象软件工程的环境中被赋予特色。领域分析过程中的步骤为:(1)定义待研究的领域;(2)对从领域中提取的项进行分类;(3)收集领域中有代表性的应用系统样本;(4)分析样本中的每个应用,并且定义分析类;(5)为这些类开发需求模型。值得注意的是,领域分析适用于任何软件工程范型,因此,领域分析可以应用到传统的软件开发和面向对象的软件开发中。

建议 这一节所讨论的分析过程主要集中于可复用构件。但是,完整的COTS系统(例如,电子商务应用、自动销售应用)分析也可以是领域分析的一部分。

13.6.2 构件的合格性检验、适应性修改与组合

领域工程提供了基于构件的软件工程(CBSE)所需的可复用构件库。某些可复用构件是自行开发的,有些可以从现有的系统中抽取得到,还可

① 第12章中曾经提到识别特定应用领域的体系结构类型。

以从第三方获得。

遗憾的是,可复用构件的存在并不能保证这些构件可以很容易或很有效地被集成到为新应用所选择的体系结构中。正因为如此,当计划使用某一构件时,要进行一系列的基于构件的开发活动。

构件合格性检验。构件合格性检验将保证某候选构件能够执行需要的功能,完全适合系统的体系结构(第12章),并具有该应用所需的质量特性(例如,性能、可靠性、可用性)。

契约式设计这种技术着重于定义明确的和可核查的构件接口规格说明,从而使构件的潜在用户快速了解其意图。我们将称为前置条件、后置条件和不变式的表述加入到构件规格说明中^①。表述使得开发人员知道构件提供什么功能,以及它在一定条件下的行为方式。这种表述使开发人员更容易识别合格的构件,因而更愿意在他们的设计中信任和使用这些构件。当构件有一个“经济型接口”时,契约式设计就得到了增强。构件接口具有一组最小的必要操作,使其能够完成职责(契约)。

接口规格说明提供了有关软件构件的操作和使用的有用信息,但是,对于确定该构件是否能在新的应用系统中高效复用,它并未提供需要的所有信息。这里列出了构件合格性检验的一些重要因素 [Bro96]:

- 应用编程接口 (Application Programming Interface, API)。
- 构件所需的开发工具与集成工具。
- 运行时需求,包括资源使用(如内存或外存储器)、时间或速度以及网络协议。
- 服务需求,包括操作系统接口和来自其他构件的支持。
- 安全特征,包括访问控制和身份验证协议。
- 嵌入式设计假设,包括特定的数值或非数值算法的使用。
- 异常处理。

提问 构件合格性的重要因素有哪些?

计划使用自行开发的可复用构件时,这些因素都是比较容易评估的。如果构件开发过程应用了良好的软件工程实践,那么之前列出的问题就都容易回答。但是,要确定商业成品构件 (Commercial Off-The-Shelf, COTS) 或第三方构件的内部工作细节就比较困难了,因为能够得到的信息可能只有接口规格说明。

构件适应性修改。在理想情况下,领域工程建立构件库,构件可以很容易地被集成到应用体系结构中。“容易集成”的含义是:(1)对于库中的所有构件,都已经实现了一致的资源管理方法;(2)所有构件都存在诸如数据管理等公共活动;(3)已经以一致的方式实现了体系结构的内部接口及与外部环境的接口。

实际上,即使已经对某个构件在应用体系结构内部的使用进行了合格性检验,也可能在刚才提到的一个或多个地方发生冲突。为了避免这些冲突,经常使用一种称为构件包装 [Bro96] 的适应性修改技术。当软件团队对某一构件的内部设计和代码具有完全的访问权时(通常不是这样,除非使用开源的 COTS 构件),则可应用白盒包装技术。与软件测试中白盒测试(第18章)相对应,白盒包装检查构件的内部处理细节,并进行代码级的

建议 软件团队除了需要评估为复用所做的适应性修改是否是成本合算的,还需要评估取得所需要的功能和性能是否也是成本合算的。

① 前置条件是对构件使用之前必须做验证的假设所作的叙述,后置条件是对将要交付的构件可提供的服务保证的叙述,不变式是对不会被构件变更的系统属性的叙述。

修改来消除所有冲突。当构件库提供了能够消除或掩盖冲突的构件扩展语言或 API 时，可以应用灰盒包装技术。黑盒包装技术需要在构件接口中引入预处理和后处理来消除或掩盖冲突。软件团队需要决定是否值得充分包装构件，或者考虑是否开发定制构件（对构件进行设计以消除遇到的冲突）。

构件组合。构件组合任务将经过合格性检验、适应性修改以及工程开发的构件组合到为应用建立的体系结构中。为完成这项任务，必须建立一个基础设施以将构件绑定到一个运行系统中。该基础设施（通常是专门的构件库）提供了构件协作的模型和使构件能够相互协作并完成共同任务的特定服务。

由于复用和 CBSE 对软件业影响巨大，因此大量的主流公司和产业协会已经提出了软件构件标准^①。这些标准包括：CCM（Corba Component Model, Corba 构件模型）^②，Microsoft COM 和 .NET^③，JavaBeans^④，以及 OSGI（Open Services Gateway Initiative 开放服务网关协议 [OSGI3]）^⑤。这些标准中没有哪一种在产业界独占优势。虽然很多开发者已经采用了其中的某个标准，但大型软件组织仍可能根据应用的类型和采用的平台来选择使用某种标准。

13.6.3 体系结构不匹配

广泛复用所面临的挑战之一就是体系结构不匹配 [Gar09a]。可复用的构件设计者常常对耦合构件的有关环境进行隐式假设。这些假设往往侧重于构件控制模型、构件的连接属性（接口）、体系结构基础设施以及体系结构构建过程中的性质。如果这些假设是不正确的，就产生了体系结构不匹配的情况。

设计概念，如抽象、隐蔽、功能独立、细化和结构化程序设计、面向对象的方法、测试、软件质量保证（SQA）和正确性验证方法，所有这些都有助于创建可复用的软件构件和防止体系结构不匹配。

如果利益相关者的设想得到了明确的记载，那么在早期就能发现体系结构不匹配。此外，风险驱动过程模型的使用强调了早期体系结构原型的定义，并且指出了不匹配的区域。如果不采取一些诸如封装或适配器^⑥的机制，往往很难修复体系结构不匹配。有时甚至需要通过完全重新设计构件接口或者通过构件本身来消除耦合问题。

13.6.4 复用的分析与设计

将需求模型（第 8～10 章）中的元素与可复用构件描述进行比较的过程有时称为“规格说明匹配” [Bel95]。如果规格说明匹配指出一个现有的构件和现有应用需求相符合，那么就可以从可复用构件库中提取这些构件，并将它们应用于新系统的设计中。如果没有发现这样的构件（即没有匹配），就需要创建新构件。在这一点上，当需要建立新构件时，应该考

① 为了实现 CBSE，过去和现在工业界都做了很多工作，Greg Olesn[Ols06] 对此给出了精彩的讨论。Ivica Crnkovic [Crb11] 给出了关于更多近期的工业构件模型的探讨。

② 关于 CCM 的进一步资料可在 www.omg.org 找到。

③ 关于 COM 和 .NET 的资料，可在 www.microsoft.com/COM 及 msdn2.microsoft.com/en-us/netframework/default.aspx 找到。

④ 关于 java bean 的最新资料可在 java.sun.com/product/javabeans/docs/ 找到。

⑤ 关于 OSGI 的资料可在 <http://www.osgi.org/Main/Homepage> 找到。

⑥ 适配器是一种软件设备，它允许具有不匹配接口的客户端访问构件，方法是将服务请求翻译成可以访问原始接口的形式。

虑可复用性设计 (Design for reuse, DFR)。

正如我们已经谈到的, DFR 需要软件工程师采用良好的软件设计概念和规则 (第 11 章)。但是, 也必须考虑应用领域的特点。Binder [Bin93] 提出了构成可复用设计基础的一系列关键问题^①。如果应用领域定义了标准的全局数据结构, 则应使用这些标准的数据结构来设计构件。在应用领域内应采用标准接口协议, 并且可选取一种适合应用领域的体系结构风格 (第 12 章) 作为新软件体系结构设计的模板。一旦建立了标准数据、接口和程序模板, 就有了进行设计所依托的框架。符合此框架的新构件将来复用的可能性就比较高。

建议 如果构件必须与遗留系统及多个系统 (它们的体系结构和接口协议不一致) 进行接口或者集成, DFR 可能会非常困难。

13.6.5 构件的分类与检索

考虑一个大型构件库, 其中存放了成千上万的可复用构件。但是, 软件工程师如何才能找到他所需要的构件呢? 为了回答这个问题, 又出现了另一个问题: 我们如何以无歧义的、可分类的术语来描述软件构件? 这些问题太难, 至今还没有明确答案。

可以用很多种方式来描述可复用软件构件, 但是理想的描述应包括 Tracz[Tra95] 提出的 3C 模型——概念 (concept)、内容 (content) 和环境 (context), 即描述构件能够实现什么功能, 如何实现那些对一般用户来讲是隐蔽的内容 (只有那些想要修改或测试该构件的人才需要了解), 以及将可复用软件构件放到什么样的应用领域中。

为了在实际环境中使用, 概念、内容和环境必须被转换为具体的规格说明模式。关于可复用软件构件分类模式的文章很多 (例如, [Nir10], [Cec06]), 所有这些分类模式都应该在可复用环境中实现, 该环境应具备以下几方面的特点:

- 能够存储软件构件和检索该构件所需分类信息的构件数据库。
- 提供访问数据库的管理系统。
- 包含软件构件检索系统 (例如对象请求代理), 允许客户应用从构件库服务器中检索构件和服务。
- 提供 CBSE 工具, 支持将复用的构件集成到新的设计或实现中。

提问 构件复用环境的关键特性是什么?

每种功能都与复用库交互, 或是嵌入在复用库中。复用库是更大型软件库 (第 21 章) 的一个元素, 并且为软件构件及各种可复用的工作产品 (例如, 规格说明、设计、模式、框架、代码段、测试用例、用户指南) 提供存储设施。

软件工具 | CBSE

[目标] 在软件构件的建模、设计、评审以及集成为更大系统的一部分时起辅助作用。

[机制] 工具的机制各异。一般情况下, CBSE 工具对以下一项或多项工作起辅助作用: 软件体系结构的规格说明和建模,

可利用的软件构件的浏览及选择, 构件集成。

[代表性工具]^②

- ComponentSource (www.componentsource.com)。提供了大量被许多不同构件

① 一般来说, DFR 准备工作应当是领域工程的一部分。

② 这里提到的工具只是此类工具的例子, 并不代表本书支持选择采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

标准支持的 COTS 软件构件（及工具）。

- Component Manager。由 Flashline (<http://www.softlookup.com/download.asp?id=8204>) 开发, “它是一个应用程序, 能支持、促进及测量软件构件复用”。
- Select Component Factory。由 Select Business Solution (www.selectbs.com) 开发, “它是一个集成的成套产品, 用于软件

设计、设计审查、服务 / 构件管理、需求管理及代码生成”。

- Software Through Pictures-ACD。由 Aonix (www.aonix.com) 发布, 对于由 OMG 模型驱动的体系结构 (一个开放的、供应商中立的 CBSE 方法), 它能够运用 UML 进行广泛的建模。

习题与思考题

- 13.1 术语“构件”有时很难定义。请首先给出一个一般的定义, 然后针对面向对象软件 and 传统软件给出更明确的定义, 最后选择三种你熟悉的编程语言来说明如何定义构件。
- 13.2 为什么传统软件当中必要的控制构件在面向对象的软件中一般是不需要的?
- 13.3 用自己的话描述 OCP。为什么创建构件之间的抽象接口很重要?
- 13.4 用自己的话描述 DIP。如果设计人员过于依赖具体构件, 会出现什么情况?
- 13.5 选择三个你最近开发的构件, 并评估每个构件的内聚类型。如果要求定义高内聚的主要优点, 那么主要优点会是什么?
- 13.6 选择三个你最近开发的构件, 并评估每个构件的耦合类型。如果要求定义低耦合的主要优点, 那么主要优点会是什么?
- 13.7 问题领域构件不会存在外部耦合的说法有道理吗? 如果你认为没有道理, 那么哪种类型的构件存在外部耦合?
- 13.8 完成: (1) 一个细化的设计类; (2) 接口描述; (3) 该类中包含的某一操作的活动图; (4) 前几章讨论过的某个 SafeHome 类的详细状态图。
- 13.9 逐步求精和重构是一回事吗? 如果不是, 它们有什么区别?
- 13.10 什么是 WebApp 构件?
- 13.11 选择已有程序的某一小部分 (大概 50 ~ 75 行源代码), 通过在源代码周围画框隔离出结构化编程构造。程序片段是否存在违反结构化程序设计原则的构造? 如果存在, 重新设计代码使其遵守结构化编程的构造, 如果不违反, 对于画出的框你注意到了什么?
- 13.12 所有现代编程语言都实现了结构化的程序设计构造。用三种程序设计语言举例说明。
- 13.13 选择有少量代码的构件, 并使用活动图来描述它。
- 13.13. 在构件级设计的评审过程中, 为什么“分块”很重要?

扩展阅读与信息资源

最近几年, 已经出版了许多有关基于构件的开发和构件复用的书籍。Szyperski (《Component Software》, 2nd ed., Addison-Wesley, 2011) 强调将软件构件作为有效系统构造块的重要性。Hamlet (《Composing Software Components》, Springer, 2010)、Curtis (《Modular Web Design》, New Riders, 2009)、Apperly 和他的同事 (《Service- and Component-Based Development》, Addison-Wesley, 2004)、Heineman 和 Councill (《Component Based Software Engineering》, Addison-Wesley, 2001)、Brown (《Large-Scale Component-Based Development》, Prentice-Hall, 2000)、Allen (《Realizing e-Business with Components》, Addison-Wesley, 2000) 以及 Leavens 和 Sitaraman (《Foundations of

Component-Based Systems》, Cambridge University Press, 2000) 编写的书覆盖了 CBSE 过程的许多重要方面。Stevens (《UML Components》, Addison-Wesley, 2006)、Apperly 和他的同事 (《Service- and Component-Based Development》, 2nd ed., Addison-Wesley, 2003) 以及 Cheesman 和 Daniels (《UML Components》, Addison-Wesley, 2000) 则侧重于用 UML 讨论 CBSE。

Malik (《Component-Based Software Development》, Lap Lambert Publishing, 2013) 提出了建立有效构件库的方法。Gross (《Component-Based Software Testing with UML》, Springer, 2010) 以及 Gao 和他的同事 (《Testing and Quality Assurance for Component-Based Software》, Artech House, 2006) 论述了基于构件的系统测试和 SQA 问题。

近些年出版了大量书籍来描述产业界基于构件的标准。这些著作既强调了关于制定标准本身的工作, 也讨论了许多重要的 CBSE 话题。

Linger、Mills 和 Witt 的著作 (《Structured Programming—Theory and Practice》, Addison-Wesley, 1979) 仍是设计方面的权威著作, 里面包含很好的 PDL, 以及关于结构化程序设计分支的细节讨论。其他书籍则集中在传统系统的过程设计问题方面, 如 Farrell (《A Guide to Programming Logic and Design》, Course Technology, 2010)、Robertson (《Simple Program Design》, 5th ed., Course Technology, 2006)、Bentley (《Programming Pearls》, 2nd ed., Addison-Wesley, 1999) 以及 Dahl (《Structured Programming》, Academic Press, 1997) 等。

最近出版的书籍中, 专门讨论构件级设计的书很少。一般来讲, 编程语言书籍或多或少关注于过程设计, 通常以书中所介绍的语言为环境进行讲解, 这方面有成百上千本书。

在网上有大量关于构件级设计的信息, 有关构件级设计的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 上找到。

用户界面设计

要点浏览

概念: 用户界面 (UI) 设计在人与计算机之间搭建了一个有效的交流媒介。遵循一系列的界面设计原则, 定义界面对象和界面动作, 然后创建构成用户界面原型基础的屏幕布局。

人员: 软件工程师通过迭代过程来设计用户界面, 这个过程采纳了被广泛接受的设计原则。

重要性: 不管软件展示了什么样的计算能力、发布了什么样的内容及提供了什么样的功能, 如果软件不方便使用、常导致用户犯错或者不利于完成目标, 你是不会喜欢这个软件的。由于界面影响用户对于软件的感觉, 因此, 它必须是令人满意的。

步骤: 用户界面设计首先要识别用户、任

务和环境需求。一旦确定用户任务, 则通过创建和分析用户场景来定义一组用户界面对象和动作。这是创建屏幕布局的基础。屏幕布局描述了图标的图形设计和位置、描述性屏幕文本的定义、窗口的规格说明和命名, 以及主要的和次要的菜单项规格说明。可以使用工具来开发原型并最终实现设计模型, 另外为了保证质量需要对结果进行评估。

工作产品: 创建用户场景, 构建产品屏幕布局, 以迭代的方式开发和修改界面原型。

质量保证措施: 原型的开发是通过用户测试驱动的, 测试驱动的反馈将用于原型的下一次迭代修改。

我们生活在充满高科技产品的世界里。几乎所有这些产品, 诸如消费电子产品、工业设备、汽车产品、企业系统、军事系统、个人计算机软件、移动 App 及 WebApp, 都需要人参与交互。如果要使一个产品取得成功, 它就必须展示出良好的可用性。可用性是指用户在使用高科技产品所提供的功能和特性时, 对使用的容易程度和有效程度的定量测量。

在计算时代的前 30 年里, 可用性并不是软件开发主要关心的。Donald Norman[Nor88] 在其关于设计的经典书籍中曾经主张 (对待可用性的) 态度改变的时机已经到来:

为了使技术适应人类, 必须要研究人类。但我们现在倾向于只研究技术。结果, 人们不得不顺从技术。而现在是时候扭转这个趋势, 使技术适应人类了。

随着技术专家对人类交互的研究, 出现了两个主要的问题。第一, 定义一组黄金规则 (14.1 节)。这些规则可以应用于所有与人交互的技术产品。第二, 定义交互机制使软件设计人员建立起可以恰当实现黄金规则的

关键概念

- 可访问性
- 命令标记
- 控制
- 设计评估
- 错误处理
- 黄金规则
- 帮助设施
- 界面分析
- 界面一致性
- 界面设计
- 界面设计模型
- 国际化
- 记忆负担
- 过程
- 响应时间

系统。这些机制消除了机器与人交互方面的一些难题，我们统一称之为用户界面。但即便是在今天，我们还是会遇到这样的用户界面：难学、难用、令人迷惑、不直观、不可原谅。在很多情况下，它们让人感到十分沮丧。然而，仍然有人在花费时间和精力去创建这样的界面，看起来，创建者并不是有意制造麻烦。

关键概念

任务分析
任务细化
可用性
用户分析

14.1 黄金规则

Theo Mandel 在其关于界面设计的著作 [Man97] 中提出了三条黄金规则：

1. 把控制权交给用户。
2. 减轻用户的记忆负担。
3. 保持界面一致。

这些黄金规则实际上构成了一系列用户界面设计原则的基础，这些原则可以指导软件设计的重要方面。

14.1.1 把控制权交给用户

在重要的、新的信息系统的需求收集阶段，曾经征求一位关键用户对窗口图形界面相关属性的意见。

该用户严肃地说：“我真正喜欢的是一个能够理解我想法的系统，它在我需要去做以前就知道我想做什么，并使我可以非常容易地完成。这就是我想要的，我也仅此一点要求。”

引述 依照用户的习惯来设计，比矫正用户的习惯要好。

Jon Meads

你的第一反应可能是摇头和微笑，但是，沉默了一会儿后，你会觉得该用户的想法绝对没有什么错。她想要一个对其要求能够做出反应并帮助她完成工作的系统。她希望去控制计算机，而不是计算机控制她。

设计者施加的大多数界面约束和限制都是为了简化交互模式。但是，这是为了谁呢？

在很多情况下，设计者为了简化界面的实现可能会引入约束和限制，其结果可能是界面易于构建，但会妨碍使用。Mandel[Man97] 定义了一组设计原则，允许用户掌握控制权。

以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式。交互模式就是界面的当前状态。例如，如果在字处理器菜单中选择拼写检查，则软件将转移到拼写检查模式。如果用户希望在这种情形下进行一些文本编辑，则没有理由强迫用户停留在拼写检查模式，用户应该能够几乎不需做任何动作就可以进入和退出该模式。

提供灵活的交互。由于不同的用户有不同的交互偏好，因此应该提供选择机会。例如，软件可能允许用户通过键盘命令、鼠标移动、数字笔、触摸屏或语音识别命令等方式进行交互。但是，每个动作并非要受控于每一种交互机制。例如，考虑使用键盘命令（或语音输入）来画一幅复杂形状的图形是有一定难度的。

允许用户交互被中断和撤销。即使陷入一系列动作之中，用户也应该能够中断动作序列去做某些其他事情（而不会失去已经做过的工作）。用户也应该能够“撤销”任何动作。

引述 我一直希望计算机能像电话一样易于使用。我的希望已经实现了，我不再想知道如何使用电话了。

Bjarne Stronstrup

当技能水平高时可以使交互流线化并允许定制交互。用户经常发现他们重复地完成相同的交互序列，因此，值得设计一种“宏”机制，使得高级用户能够定制界面以方便交互。

使用户与内部技术细节隔离开来。用户界面应该能够将用户移入应用的虚拟世界中，用户不应该知道操作系统、文件管理功能或其他隐秘的计算技术。

设计应允许用户与出现在屏幕上的对象直接交互。当用户能够操纵完成某任务所必需的对象，并且以一种该对象好像是真实存在的方式来操纵它时，用户就会有一种控制感。例如，允许用户将文件拖到“回收站”的应用界面，即是直接操纵的一种实现。

14.1.2 减轻用户的记忆负担

一个经过精心设计的用户界面不会加重用户的记忆负担，因为用户必须记住的东西越多，和系统交互时出错的可能性也就越大。只要可能，系统应该“记住”有关的信息，并通过有助于回忆的交互场景来帮助用户。Mandel[Man97]定义了一组设计原则，使得界面能够减轻用户的记忆负担。

减少对短期记忆的要求。当用户陷于复杂的任务时，短期记忆的要求会很强烈。界面的设计应该尽量不要求记住过去的动作、输入和结果。可行的解决办法是通过提供可视的提示，使得用户能够识别过去的动作，而不是必须记住它们。

建立有意义的默认设置。初始的默认集合应该对一般的用户有意义，但是，用户应该能够说明个人的偏好。然而，“重置”（reset）选项应该是可用的，使得可以重新定义初始默认值。

定义直观的快捷方式。当使用助记符来完成系统功能时（如用 Alt+P 激活打印功能），助记符应该以容易记忆的方式联系到相关动作（例如，使用要激活任务的第一个字母）。

界面的视觉布局应该基于真实世界的象征。例如，一个账单支付系统应该使用支票簿和支票登记簿来指导用户的账单支付过程。这使得用户能够依赖于很好理解的可视化提示，而不是记住复杂难懂的交互序列。

以一种渐进的方式揭示信息。界面应该以层次化方式进行组织，即关于某任务、对象或行为的信息应该首先在高抽象层次上呈现。更多的细节应该在用户表明兴趣后再展示。

SafeHome 违反用户界面的黄金规则

[场景] Vinod 的工作间，用户界面设计启动在即。

[人物] Vinod 和 Jamie，SafeHome 软件工程团队成员。

[对话]

Jamie: 我已经在考虑监控功能的界面了。

Vinod (微笑): 思考是好事。

Jamie: 我认为我们可以将其简化。

Vinod: 什么意思？

Jamie: 如果我们完全忽略住宅平面图会怎么样？它倒是很华丽，但是会带来很多开发工作量。我们只要询问用户要查看的指定摄像机，然后在视频窗口显示视频就

可以了。

Vinod: 房主如何记住有多少个摄像机以及它们都安装在什么地方呢？

Jamie (有点不高兴): 他是房主，应该知道。

Vinod: 但是如果不知道呢？

Jamie: 应该知道。

Vinod: 这不是问题的关键……如果忘记了呢？

Jamie: 哦，我应该提供一张可操作的摄像机及其位置的清单。

Vinod: 那也有可能，但是为什么要有一份清单呢？

Jamie: 好的，无论用户是否有这方面的

要求，我们都提供一份清单。

Vinod：这样更好。至少用户不必特意记住我们给他的东西了。

Jamie（想了一会儿）：但是你喜欢住宅平面图，不是吗？

Vinod：哈哈。

Jamie：你认为市场营销人员会喜欢哪一个？

Vinod：你在开玩笑，是吗？

Jamie：不。

Vinod：哦……华丽的那个……他们喜欢迷人的……他们对简单的不感兴趣。

Jamie：（叹口气）好吧，也许我应该为两者都设计一个原型。

Vinod：好主意……我们就让客户来决定。

14.1.3 保持界面一致

用户应该以一致的方式展示和获取信息，这意味着：（1）按照贯穿所有屏幕显示的设计规则来组织可视信息；（2）将输入机制约束到有限的集合，在整个应用中得到一致的使用；（3）从任务到任务的导航机制要一致地定义和实现。Mandel[Man97]定义了一组帮助保持界面一致性的设计原则。

允许用户将当前任务放入有意义的环境中。很多界面使用数十个屏幕图像来实现复杂的交互层次。提供指示器（例如，窗口标题、图标、一致的颜色编码）帮助用户知晓当前工作环境是十分重要的。另外，用户应该能够确定他来自何处以及存在哪些转换到新任务的途径。

在完整的产品线内保持一致性。一个应用系列（即一个产品线）都应采用相同的设计规则，以保持所有交互的一致性。

如果过去的交互模型已经建立起了用户期望，除非有不得已的理由，否则不要改变它。一个特殊的交互序列一旦变成事实上的标准（如使用 Alt+S 来存储文件），则用户在遇到每个应用时均会如此期望。如果改变这些标准（如使用 Alt+S 来激活缩放比例），将导致混淆。

本节和前面几节讨论的界面设计原则为软件工程师提供了基本指南。在下面几节中，我们将考察界面设计过程。

引述 看起来不同的事物产生的效果应该不同，而看起来相同的事物产生的效果应该相同。

Larry Marine

信息栏 可用性

在一篇关于可用性的见解深刻的论文中，Larry Constantine[Con95]提出了一个与可用性主题非常相关的问题：“用户究竟想要什么？”他给出了下面的回答。

“用户真正想要的是好的工具。所有的软件系统，从操作系统和语言到数据录入和决策支撑应用软件，都是工具。最终用户希望从为其设计的工具中得到的东西，与我们希望从所使用工具中得到的是

一样的。他们想要易于学习并能够为自己工作提供帮助。同时，他们想要的系统应该能提高工作效率，不会欺骗或困扰他们，不会使他们易于犯错误或难于完成工作。”

Constantine 指出，系统的可用性并非取决于设计美学、交互技术的发展水平或者内置的界面智能等方面，而是当界面的架构适合于将要使用这些界面的用户的需

求时,才能获得可用性。

正式的可用性定义往往令人有些迷惑。Donahue 和他的同事 [Don99] 给出了如下的定义:“可用性是一种衡量计算机系统好坏的度量……便于学习;帮助初学者记住他们已经学到的东西;降低犯错的可能;使得用户更加有效率,并且使得他们对系统感到满意。”

确定你所建系统是否可用的唯一办法就是进行可用性评估和测试。观察用户与系统的交互,同时回答下列问题 [Con95]:

- 在没有连续的帮助或用法说明的情况下,系统是否便于使用?
- 交互规则是否能够帮助一个知识渊博的用户工作得更加有效率?
- 随着用户的知识不断增多,交互机制是否能变得更灵活?
- 系统是否已经过调试,使之适应其运行的物理环境和社会环境?

- 用户是否意识到系统的状态?在工作期间,用户是否能够知道其所处的位置?
- 界面是否是按照一种合理并且一致的方式来构建的?
- 交互机制、图标和过程是否在整个界面中一致?
- 交互是否能够提前发现错误并帮助用户修正它们?
- 界面是否能够容错?
- 交互是否简单?

如果上述每个问题的回答都是肯定的,那么可以认为这个系统是可用的。

可用性好的系统带来的诸多好处在于 [Don99]: 提高销售量和用户满意度、具有竞争优势、在媒体中获得良好的评价、获得良好的口碑、降低支持成本、提升最终用户生产力、降低培训费用、减少文档开销、减少来自不满意用户的投诉。

14.2 用户界面的分析和设计

用户界面的分析和设计全过程始于创建不同的系统功能模型(从外部看时对系统的感觉)。首先将完成系统功能的任务分为面向人的和面向计算机的,然后考虑那些应用到界面设计中的各种设计问题。可以使用各种工具来建造原型并最终实现设计模型,最后由最终用户从质量的角度对结果进行评估。

14.2.1 用户界面分析和设计模型

分析和设计用户界面时要考虑四种模型:工程师(或者软件工程师)建立用户模型;软件工程师创建设计模型;最终用户在脑海里对界面产生映像,称为用户的心理模型或系统感觉;系统的实现者创建实现模型。不幸的是,这四种模型可能相差甚远。界面设计人员的任务就是消解这些差距,导出一致的界面表示。

用户模型确立了系统最终用户的轮廓(profile)。Jeff Patton[Pat07]在《用户为中心的设计》(user-centric design)的前言中写道:

事实是,设计者和开发者(包括我自己)都经常考虑到用户。然而,在缺少特定用户有力的心理模型的情况下,开发者和设计者会以自我来替代用户。自我替代并不是用户为中心,而是自我为中心。

网络资源 可以在 www.nngroup.com 找到用户界面设计信息的优秀资源。

引述 如果用户界面有一点瑕疵,那么整个用户界面都会被破坏。

Douglas
Anderson

为了建立有效的用户界面，“开始设计之前，必须对预期用户加以了解，包括年龄、性别、身体状况、教育、文化和种族背景、动机、目标以及性格”[Shn04]。此外，可以将用户分类：新手，对系统有了解的用户，间歇用户或对系统有了解的经常用户。

用户的心理模型（系统感觉）是最终用户在脑海里对系统产生的印象，例如，请某个餐厅评级移动 App 的用户来描述其操作，那么系统感觉将会引导用户的回答，准确的回答取决于用户的经验（新手只能做简要的回答）和用户对应用领域软件的熟悉程度。一个对餐厅评级应用程序有深刻了解但只使用这种系统几次的用户，可能比已经使用该系统好几个星期的新手对该应用程序的功能描述回答得更详细。

实现模型组合了计算机系统的外在表现（界面的观感），结合了所有用来描述系统语法和语义的支撑信息（书、手册、录像带、帮助文件）。当系统实现模型和用户心理模型相一致的时候，用户通常就会对软件感到很舒服，使用起来就很有效。为了将这些模型融合起来，所开发的设计模型必须包含用户模型中的一些信息，实现模型必须准确地反映界面的语法和语义信息。

建议 即使用户是新手也会有使用快捷键的需求；即使是经常使用系统的用户有时候也需要指导。他们的要求都要满足。

引述 注意用户的行为而不是他们的言语。

Jakob Nielsen

14.2.2 过程

用户界面的分析和设计过程是迭代的，可以用类似于第 4 章讨论过的螺旋模型表示。如图 14-1 所示，用户界面分析和设计过程开始于螺旋模型的内部，且包括四个不同的框架活动 [Man97]：（1）界面分析和建模；（2）界面设计；（3）界面构建；（4）界面确认。图 14-1 中的螺旋意味着每个活动都将多次出现，每绕螺旋一周表示需求和设计的进一步细化。在大多数情况下，构建活动涉及原型开发——这是唯一实用的确认设计结果的方式。

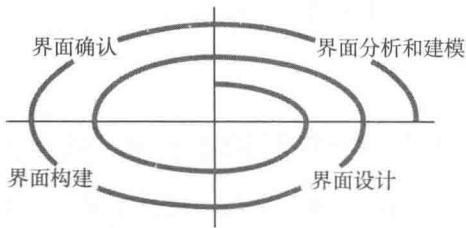


图 14-1 用户界面的设计过程

界面分析活动的重点在于那些与系统交互的用户的轮廓。记录技能级别、业务理解以及对新系统的一般感悟，并定义不同的用户类别。对每个用户类别进行需求引导。本质上，软件工程师试图去理解每类用户的系统感觉（14.2.1 节）。

一旦定义好了一般需求，就将进行更详细的任务分析。标识、描述和细化（通过绕螺旋的多次迭代）用户为了达到系统目标而执行的任务。14.3 节将对任务分析进行更详细的讨论。最后，用户环境的分析着重于物理工作环境的特征（例如地理位置、采光、位置约束）。

作为分析动作的一部分而收集的信息被用于创建界面的分析模型。使用该模型作为基础，设计活动便开始了。

界面设计的目标是定义一组界面对象和动作（以及它们的屏幕表示），使得用户能够以满足系统所定义的每个使用目标的方式完成所有定义的任务。界面设计将在 14.4 节详细讨论。

界面构建通常开始于创建可评估使用场景的原型。随着迭代设计过程的继续，用户界面开发工具可用来完成界面的构造。

界面确认着重于：（1）界面正确地实现每个用户任务的能力，适应所有任务变化的能力以及达到所有一般用户需求的能力；（2）界面容易使用和学习的程度；（3）作为工作中的得

力工具，用户对界面的接受程度。

如我们已经提到的，本节描述的活动是以迭代方式开展的。因此，不需要在第一轮就试图刻画所有的细节（对分析或设计模型而言）。后续的过程将细化界面的任务细节、设计信息和运行特征。

14.3 界面分析^①

所有软件工程过程模型的一个重要原则是：在试图设计一个解决方案之前，最好对问题有所理解。在用户界面的设计中，理解问题就意味着了解：（1）通过界面和系统交互的人（最终用户）；（2）最终用户为完成工作要执行的任务；（3）作为界面的一部分而显示的内容；（4）任务处理的环境。在接下来的几节中，为了给设计任务建立牢固的基础，我们来检查界面分析的每个成分。

14.3.1 用户分析

在担忧技术上的问题之前，用户界面这个词完全有理由要求我们花时间去理解用户。之前，我们提到每个用户对于软件都存在心理映像，而这可能与其他用户的心理映像存在着差别。另外，用户的心理映像可能与软件工程师的设计模型相距甚远。设计师能够将得到的心理映像和设计模型聚合在一起的唯一办法就是努力了解用户，同时了解这些用户是如何使用系统的。为了完成这个任务，可以利用各种途径（用户访谈、销售输入、市场输入、支持输入）获得的信息。

下列一组问题（改编自 [Hac98]）将有助于界面设计师更好地理解系统的用户：

- 用户是经过训练的专业人员、技术员、办事员，还是制造业工人？
- 用户平均正规教育水平如何？
- 用户是否具有学习书面资料的能力或者是否渴望接受集中培训？
- 用户是专业录入人员还是键盘恐惧者？
- 用户群体的年龄范围如何？
- 是否需要考虑用户的性别差异？
- 如何为用户完成的工作提供报酬？
- 用户是否在正常的办公时间内工作或者一直干到工作完成？
- 软件是用户所完成工作中的一个集成部分，还是偶尔使用一次？
- 用户群中使用的主要交流语言是什么？
- 如果用户在使用软件的过程中出错，结果会怎么样？
- 用户是否是系统所解决问题领域的专家？
- 用户是否想了解界面背后的技术？

提问 我们如何知道最终用户的人数和特征？

这些问题和类似问题的答案将帮助设计师了解：最终用户是什么人，什么可能令他们感到愉悦，如何对用户进行分类，他们对系统的心理模型是什么样子，用户界面必须具有哪些特性才能满足用户的需求。

^① 因为需求分析问题在第7~10章已经讨论过，所以有理由把这一节放到这几章中去。本节之所以放在这里，是因为界面的分析和设计紧紧相连，两者的界限常常模糊不清。

14.3.2 任务分析和建模

任务分析的目标就是给出下列问题的答案：

- 在指定环境下用户将完成什么工作？
- 用户工作时将完成什么任务和子任务？
- 在工作中用户将处理什么特殊的问题域对象？
- 工作任务的顺序（工作流）如何？
- 任务的层次关系如何？

为了回答这些问题，软件工程师必须利用本书前面所讨论的分析技术，只不过在此种情况下，要将这些技术应用到用户界面。

用例。在前面几章中，我们提到过用例描述了参与者（在用户界面设计中，参与者通常是某个人）和系统的交互方式。作为任务分析的一部分，设计用例用来显示最终用户如何完成指定的相关工作任务。在大多数情况下，用例采用第一人称并以非正式形式（一段简单的文字）来书写。例如，假如一家小的软件公司想专门为公司室内设计师开发一个计算机辅助设计系统。为了更好地理解他们是如何工作的，实际的室内设计师应该描述特定的设计功能。在室内设计师被问到“如何确定室内家具摆放位置”的时候，室内设计师写下了如下非正式的用例描述：

我从勾画房间的平面图、窗户与门的尺寸和位置开始设计。我非常关心射入房间的光线，关心窗外的风景（如果它很漂亮，就会吸引我的注意力），关心无障碍墙的长度，关心房间内活动空间的通道大小。我接下来会查看客户和我选取的家具清单……接着，我会为客户画出一个房屋的透视图（三维图画），让客户感受到房间看起来应该是什么样的。

这个用例给出了计算机辅助设计系统中一项重要工作任务的基本描述。从这个描述中，软件工程师能够提炼出任务、对象和整个交互流程。另外，系统中能够使得室内设计师感到愉悦的其他特征也被构思出来。例如，可以将房屋中每一扇窗户的风景都拍摄成一张数码相片。在画房屋透视图时，通过每扇窗户就可以看到窗外的真实景象。

关键点 用户的目的是通过用户界面来完成一个或多个任务。为了实现这一点，用户界面必须提供用户达到目标的机制。

网络资源 可以在 <http://web.ee.cs.umich.edu/~kieras/docs/GOMS/> 找到不错的用户建模信息资源。

SafeHome 用户界面设计的用例

[场景] Vinod 的工作间，用户界面设计正在进行。

[人物] Vinod 和 Jamie，SafeHome 软件工程师团队成员。

[对话]

Jamie：我拦住我们的市场部联系人，让她写了一份监视界面的用例。

Vinod：站在谁的角度来写？

Jamie：当然是房主，还会有谁？

Vinod：还有系统管理员这个角色。即使是房主担任这个角色，这也是一个不同的视角。“管理员”启动系统，配置零件，

布置平面图，安置摄像机……

Jamie：当房主想看视频时，我只是让她扮演房主的角色。

Vinod：好的，这只是监视功能界面主要行为之一。但是，我们也应该调查一下系统管理员的行为。

Jamie（有些不悦）：你是对的。

（Jamie 离开去找销售人员。几个小时以后她回来了。）

Jamie：我真走运，找到了市场部联系人，我们一起完成了系统管理员的用例。我们应该把“管理”定义为可以应用所有其他

SafeHome 功能的一个功能。这是我们提出的用例。

(Jamie 给 Vinod 看这个非正式的用例。)

非正式用例：我想能够在任何时候设置和编辑系统的布置方案。当我启动系统时，我选择某个管理功能。系统询问我是否要建立一个新的系统布置方案，或者询问我是否编辑已有的方案。如果我选择了一个新建方案，系统呈现一个绘画屏幕，在网格上可以画出建筑平面图来。为了绘画简便，应该提供墙壁、窗户和门的图标。我只是将图标伸展到合适的长度。系统将把长度显示为英尺或者米（我可以选择度量系统）。我能够从传感器和摄像机库中进行选择，并且将它们放置在平面图中。我

标记每个传感器和摄像机，或者系统自动进行标记。我可以通过合适的菜单对传感器和摄像机进行设置。如果选择编辑，就可以移动传感器和摄像机，添加新的或删除已有的传感器和摄像机，编辑平面图并编辑摄像机和传感器的设置。在每种情形下，我希望系统能够进行一致性检查并且帮助我避免出错。

Vinod (看完脚本之后)：好的，对于绘画程序，可能有一些有用的设计模式（第11章）或可复用的图形用户界面构件。我打赌，通过使用可复用构件，我们可以实现某些或大部分管理员界面。

Jamie：同意！我马上进行检查。

任务细化。在第11章中，我们讨论了逐步求精（也称为功能分解或者逐步细化），把它作为一种细化处理任务的机制，而这些任务是软件完成某些期望功能所要求的。界面设计的任务分析采用了一种详细阐述的办法来辅助理解用户界面必须采纳的用户活动。

首先，工程师必须定义完成系统或应用程序目标所需的任务并对任务进行划分。例如，考虑前面讨论的为室内设计师开发的计算机辅助设计系统。通过观察工作中的室内设计师，软件工程师了解到，室内设计由一系列的主要活动组成：家具布置（在前面用例设计中提到过）、结构和材料的选择、墙壁和窗户装饰物的选择、（向客户）展示、计算成本、购物。其中任何一个都可以被细化成一系列的子任务。例如，使用用例中的信息，可以将家具布置任务细化为下面的子任务：（1）根据房屋的尺寸画出平面图；（2）将门窗安置在合适的位置；（3a）使用家具模型在平面图上描绘相应比例的家具轮廓；（3b）使用饰件模板在平面设计图上勾勒相应比例的饰件；（4）移动家具和饰件轮廓线到达理想的位置；（5）标记所有的家具和饰件轮廓；（6）标出尺寸以显示其位置；（7）为用户勾画透视图。也可以应用类似的方法对其他主任务进行细化。

上面的每个子任务都可以进一步细化。其中子任务1~6可以通过在界面中操纵信息和执行各种动作来完成。另一方面，子任务7可以在软件中自动完成，并且几乎不用直接为用户交互^①。界面的设计模型应该以一种与用户模型（典型室内设计师的轮廓图）和系统感觉（室内设计师期望系统自动提供）相一致的方式来配合这些任务。

对象细化。软件工程师这时不是着眼于用户必须完成的任务，而是需要检查用例和来自用户的其他信息，并且提取室内设计师需要使用的物理对象。这些对象可以分为不同的类。需要定义每个类的属性，并且通过对

建议 尽管对象细化十分有用，但它应当作为独立的方法去使用。任务分析的过程中，应当考虑用户的声。

① 然而，事实可能不是这样。室内设计师可能想要指定所画的透视图、缩放比例、色彩的运用和其他信息。与透视渲染相关的用例将提供解决这些问题的信息。

每个对象动作的评估为设计师提供一个操作列表。例如，家具模板可能被转换成一个名为 Furniture 的类，这个类包括 size、shape 和 location 等属性。室内设计师会从 Furniture 类中选择对象，将其移动到平面图（在此处，平面图是另一个对象）中的某个位置上，拖曳家具的轮廓，依此类推。任务选择（select）、移动（move）、拖曳（draw）等都是操作。用户界面分析模型不能对任何一种操作都提供文字实现。然而，随着设计的不断细化，对每个操作的细节都会进行定义。

工作流分析。当大量扮演着不同角色的用户使用某个用户界面时，有时候除了任务分析和对象细化之外，还有必要进行工作流分析。该技术使得软件工程师可以很好地理解在涉及多个成员（角色）时，工作过程是如何完成的。假设某个公司打算将处方药的开方和给药过程全部自动化。全部过程^①将围绕着一个 WebApp 进行考虑，医生（或者他们的助手）、药剂师和病人等都可以访问这个应用系统。用 UML 泳道图（活动图的一种变形）能够有效地表示工作流。

下面只考虑工作过程中的一小部分：当病人请求重填处方时发生的情形。图 14-2 给出了一个泳道图，该图表明了前面提及的三个角色的任务和决定。这些信息可以通过访谈或每个角色书写的用例获取。不管怎样，事件流（图中显示的）使得界面设计师认识到三个关键的界面特征：

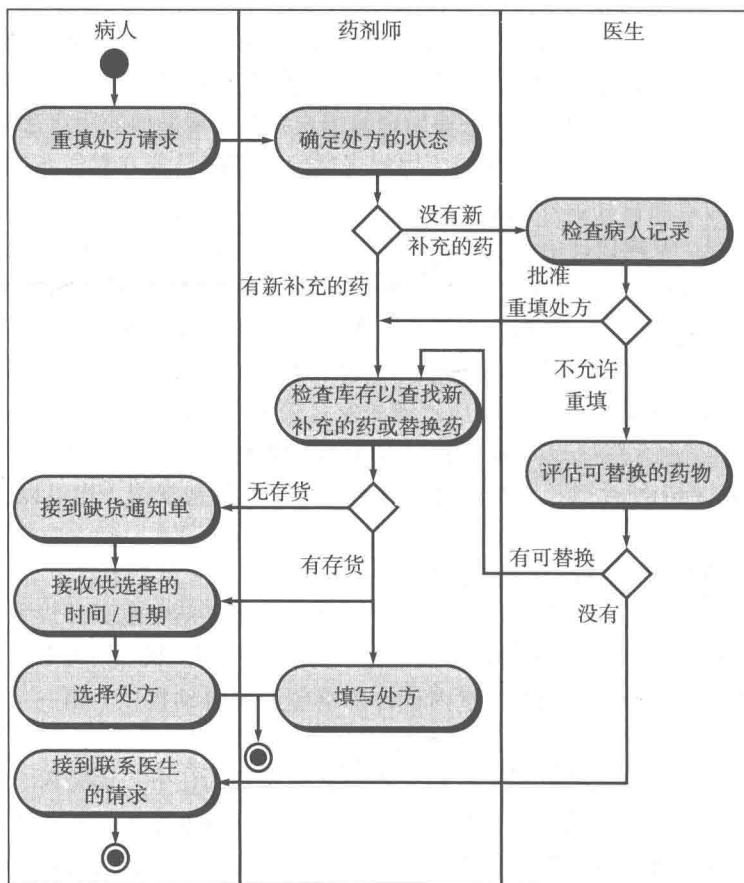


图 14-2 处方重填功能的泳道图

① 这个例子选自 [Hac98]。

1. 每个用户通过界面实现不同的任务，因此，为病人设计的界面在感官上与为药剂师或医生设计的界面有所不同。
2. 为医生和药剂师设计的界面应该能够访问和显示来自辅助信息源的信息（例如，药剂师应能够访问库存详细清单，而医生应能够访问其他可选药物信息）。
3. 泳道图中的很多活动都可以采用任务分析和对象求精使其进一步细化（例如，“填写处方”隐含着邮购支付、访问药房，或者访问特殊药品分发中心）。

引述 使技术适应用户要比用户适应技术好。

Larry Marine

层次表示。在界面分析时，会产生相应的细化过程。一旦建立了工作流，就为每个用户类型都定义一个任务层次。该任务层次来自于为用户定义的每项任务的逐步细化。例如，考虑重填处方的用户任务请求，任务层次如下：

重填处方请求

- 提供辨识信息
 - 指定姓名
 - 指定用户 ID
 - 指定个人身份识别号码 (PIN) 和密码
- 指定处方序号
- 指定重填处方所需的日期

为了完成填写处方的任务，定义了三个子任务。可以将其中的第一个子任务“提供辨识信息”进一步细化成三个另外的子任务。

14.3.3 显示内容分析

14.3.2 节中标识出的用户任务导致需要对各种各样不同类型的内容进行描述。在第 8 章和第 10 章中讨论过的分析建模技术标识出由应用产生的输出数据对象。这些数据对象可能：（1）由应用中其他部分的构件（与界面无关）生成；（2）由应用所访问数据库中存储的数据获得；（3）从系统外部传递到正在讨论的应用中。

在界面分析步骤中，要考虑内容的格式和美感（当它要显示在界面上时）。其中需要提问和回答的问题包括：

- 不同类型的数据是否要放置到屏幕上固定的位置（例如，照片一般显示在右上角）？
- 用户能否定制内容的屏幕位置？
- 是否对所有内容赋予适当的屏幕标识？
- 为了便于理解，应如何划分长篇报告？
- 对于大集合的数据，是否存在直接移动到摘要信息的机制？
- 输出图形的大小是否需要适合所使用显示设备的限制？
- 如何使用颜色来增强理解？
- 出错信息和警告应如何呈现给用户？

提问 作为用户界面设计的一部分，我们如何决定所显示内容的格式和美感？

对这些（和其他）问题的回答有助于软件工程师建立起内容表示的需求。

14.3.4 工作环境分析

Hackos 和 Redish[Hac98] 在讨论工作环境分析的重要性时这样写道：“人们不能孤立地

完成任务。他们会受到周围活动的影响,如工作场所的物理特征,使用设备的类型,与其他人的工作关系等。”在某些应用中,计算机系统的用户界面被放在“用户友好”的位置(例如,合适的亮度、良好的显示高度、简单方便的键盘操作),但有些地方(例如,工厂的地板和飞机座舱)亮度可能不是很适合,噪音也可能是个问题,也许不能选择使用键盘、鼠标或触摸屏,显示方位也不甚理想。界面设计师可能会受到某些因素的限制,这些因素会减弱易用性。

除了物理的环境因素之外,工作场所的文化氛围也起着作用。可否采用某种方式(例如,每次交互所用时间、交互的准确性)来度量系统的交互?在提供一个输入前,两个或多个人员是否一定要共享信息?如何为系统用户提供支持?在界面设计开始之前,应该对上述问题和更多的相关问题给予回答。

14.4 界面设计步骤

一旦完成了界面分析,最终用户要求的所有任务(对象和动作)都被详细确定下来,界面设计活动就开始了。与所有的软件工程设计一样,界面设计是一个迭代的过程。每个用户界面设计步骤都要进行很多次,每次精细化的信息都来源于前面的步骤。

尽管已经提出了很多不同的用户界面设计模型(例如[Nor86]和[Nie00]),但它们都建议结合以下步骤:(1)定义界面对象和动作(操作);(2)确定事件(用户动作),即会导致用户界面状态发生变化的事件;(3)描述每个状态的表示形式;(4)说明用户如何利用界面提供的信息来解释每个状态。

引述 交互设计是图形艺术、技术和心理学的无缝结合。

Brad Wieners

14.4.1 应用界面设计步骤

界面设计的一个重要步骤是定义界面对象和作用于对象上的动作。为了完成这个目标,需要使用类似于第8章介绍的方法来分析用户场景,也就是说,撰写用例的描述。名词(对象)和动词(动作)被分离出来形成对象和动作列表。

一旦完成了对象和动作的定义及迭代细化,就可以将它们按类型分类。目标、源和应用对象都被标识出来。将源对象(如报告图标)拖放到目标对象(如打印机图标)上,这意味着该动作要产生一个硬拷贝的报告。应用对象代表应用中特有的数据,它们并不作为屏幕交互的一部分被直接操纵。例如,邮件列表被用于存放邮件的名字,该列表本身可以进行排序、合并或清除(基于菜单的动作),但是,它不会通过用户的交互被拖动和删除。

当设计者满意地认为已经定义了所有的重要对象和动作(对一次设计迭代而言)时,便可以开始进行屏幕布局。与其他界面设计活动一样,屏幕布局是一个交互过程,其中包括:图标的图形设计和放置、屏幕描述性文字的定义、窗口的规格说明和标题,以及各类主要和次要菜单项的定义等。如果一个真实世界的隐喻适合于该应用,则在此时进行说明,并以补充隐喻的方式来组织布局。

为了对上面的设计步骤提供简明的例证,我们考虑 SafeHome 系统(在前面几章讨论过的一个用户场景。下面是界面的初步用例(由房主写的)描述。

初步用例:我希望通过 Internet 在任意的远程位置都能够访问 SafeHome 系统。使用运行在笔记本上的浏览器软件(当正处于工作或者旅行状态时),我可以决定报警系统的状态、启动或关闭系统、重新配置安全区以及通过预先安置的摄像机观察住宅内的不同房间。

为了远程访问 SafeHome，我需要提供标识符和密码，这些定义了访问的级别（如并非所有用户都可以重新配置系统）并提供安全保证。一旦确认了身份，我就可以检查系统状态，并通过启动或关闭 SafeHome 系统改变状态。通过显示住宅的平面图，观察每个安全传感器，显示每个当前配置区域以及修改区域（必要时），可以重新配置系统。通过有策略地放置摄像机以观察房子内部。通过对每个摄像机进行摇动和变焦以提供房子内部的不同视角。

基于这个用例，确定房主的任务、对象和数据项如下：

- 访问 SafeHome 系统。
- 输入 ID 和密码实现远程访问。
- 检查系统状态。
- 启动或关闭 SafeHome 系统。
- 显示平面图和传感器位置。
- 显示平面图上的区域。
- 改变平面图上的区域。
- 显示建筑平面图上的视频摄像机位置。
- 选择用于观察的视频摄像机。
- 观察视频图像（每秒 4 帧）。
- 摇动或变焦摄像机。

从房主的这个任务清单中抽取出对象和动作。所提到的大部分对象都是应用对象。然而，视频摄像机位置（源对象）被拖放到视频摄像机（目标对象）以创建视频图像（视频显示的窗口）。

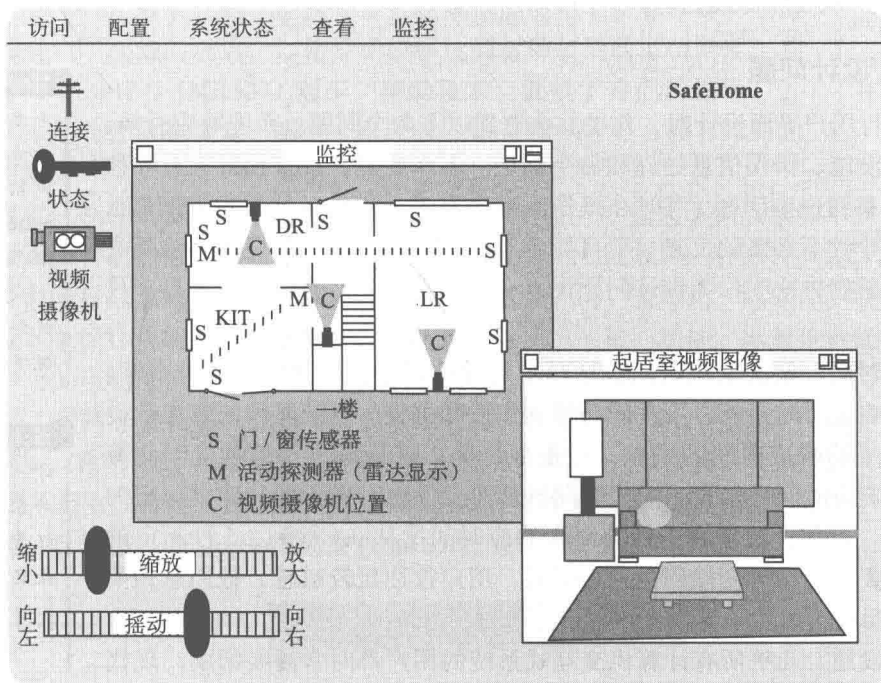


图 14-3 基本的屏幕布局

为视频监控设计的屏幕布局初步草图如图 14-3 所示^①。为了调用视频图像,需选择显示在监控窗口中的建筑平面图上的视频摄像机位置图标 C。在这种情况下,起居室(LR)中的摄像机位置被拖放到屏幕左上部分的视频摄像机图标处,此时,视频图像窗口出现,显示来自位于起居室中的摄像机的流视频。变焦和摇动控制条用于控制视频图像的放大和方向。为了选择来自另一个摄像机的图像,用户只需简单地将另一个不同的摄像机位置图标拖放到屏幕左上区域的摄像机图标上即可。

所显示的布局草图需以菜单条上每个菜单项的扩展来补充,指明视频监控模式(状态)有哪些可用的动作。在界面设计过程中,将创建用户场景中提到的房主的每个任务的一组完整草图。

14.4.2 用户界面设计模式

图形用户界面已经变得如此普遍,以至于涌现出各式各样的用户界面设计模式。设计模式是一种抽象,描述了特定的、界限明确的设计问题的解决方案。

作为通常碰到的界面设计问题的一个例子,考虑用户必须一次或多次输入日历日期这种情况,有时候需要提前输入月份。对于这个简单的问题,有很多可能的解决方案,为此也提出了很多种不同的模式。Laakso[Laa00]提出了一种称为 CalendarStrip 的模式,此模式生成一个连续、滚动的日历,在这个日历上,当前日期被高亮度显示,未来的日期可以在日历上选择。这个日历隐喻在用户中具有很高的知名度,并提供了一种有效的机制,可以在上下文环境中设置未来的日期。

在过去的十年间,人们已经提出了很多用户界面设计模式。此外,Erickson[Eri08]提供了许多基于 Web 的文献资料。

14.4.3 设计问题

在进行用户界面设计时,几乎总会遇到以下四个问题:系统响应时间、用户帮助设施、错误信息处理和命令标记。不幸的是,许多设计人员往往很晚才注意到这些问题(有时在操作原型已经建立起来后才发现问题),这往往会导致不必要的反复、项目拖延及用户的挫折感,最好的办法是在设计的初期就将这些作为设计问题加以考虑,因为此时修改比较容易,代价也低。

响应时间。系统响应时间包括两个重要的属性:时间长度和可变性。如果系统响应时间过长,用户就会感到焦虑和沮丧。系统时间的可变性是指相对于平均响应时间的偏差,在很多情况下这是最重要的响应时间特性。即使响应时间比较长,响应时间的低可变性也有助于用户建立稳定的交互节奏。例如,稳定在 1 秒的命令响应时间比从 0.1 秒到 2.5 秒不定的响应时间要好。在可变性到达一定值时,用户往往比较敏感,他们总是关心界面背后是否发生了异常。

帮助设施。几乎所有计算机交互式系统的用户都时常需要帮助。现代

建议 尽管自动化的工具在开发布局原型中十分有用,但有时候铅笔和纸张也是需要的。

网络资源 已经有大量的用户界面设计模式,访问 <http://www.hci-patterns.org/patterns/borchers/patternindex.html> 可以找到它们的站点链接。

引述 当试图设计一些十分简单的东西时,人们经常犯的共性错误就是低估了笨人的智慧。

Douglas Adams

^① 注意这里的实现与前几章讲到的这些特性的实现有所不同。这里应该是第一次设计的草图,可以考虑提供备选的设计草图。

的软件均提供联机帮助，用户可以不离开用户界面就解决问题。

错误处理。通常，交互式系统给出的出错消息和警告应具备以下特征：（1）以用户可以理解的语言描述问题；（2）应提供如何从错误中恢复的建设性意见；（3）应指出错误可能导致哪些不良后果（比如破坏数据文件），以便用户检查是否出现了这些情况（或者在已经出现的情况下进行改正）；应伴随着视觉或听觉上的提示，并且永远不应该把错误归咎于用户。

菜单和命令标记。键入命令曾经是用户和系统交互的主要方式，并广泛用于各种应用。现在，面向窗口的界面采用点击（point）和选取（pick）方式，减少了用户对键入命令的依赖。但许多高级用户仍然喜欢面向命令的交互方式。在提供命令或菜单标签交互方式时，必须考虑以下问题：

- 每个菜单选项是否都有对应的命令？
- 以何种方式提供命令？有三种选择：控制序列（如 Alt+P）、功能键或键入命令。
- 学习和记忆命令的难度有多大？命令忘了怎么办？
- 用户是否可以定制和缩写命令？
- 在界面环境中菜单标签是否是自解释的？
- 子菜单是否与主菜单项所指功能相一致？
- 有适合于应用系列内部的命令使用约定吗？

应用的可访问性。随着计算型应用变得无处不在，软件工程师必须确保界面设计中包含使得有特殊要求的用户易于访问的机制。对于那些实际上面临挑战的用户（和软件工程师）来说，由于道义、法律和业务等方面的原因，可访问性是必需的。有多种可访问性指导方针（如 [W3C03]）——很多都是为 WebApp 设计的，但这些方针经常也能应用于所有软件——为设计界面提供了详细的建议，以使界面能够达到各种级别的可访问性。其他指南（如 [App13]、[Mic13]）对于“辅助技术”提供了专门的指导，这些技术用来解决那些在视觉、听觉、活动性、语音和学习等方面有障碍的人员的需要。

国际化。软件工程师和他们的经理往往会低估建立一个适应不同国家和不同语言需要的用户界面所应付出的努力和技能。用户界面经常是为一个国家和一种语言所设计的，在面对其他国家时只好应急对付。设计师面临的挑战就是设计出“全球化”的软件。也就是说，用户界面应该被设计成能够容纳需要交付给所有软件用户的核心功能。本地化特征使得界面能够针对特定的市场进行定制。

软件工程师有多种国际化指导方针（如 [IBM13]）可以使用。这些方针解决了宽度设计问题（例如，在不同的市场情况下屏幕布局可能是不同的），以及离散实现问题（例如，不同的字母表可能生成特定的标识和间距需求）。对于几十种具有成百上千字母和字符的自然语言的管理，已经提出的 Unicode 标准 [Uni03] 就是用来解决这个挑战性问题的。

引述 来自地狱的界面——修正这个错误并且继续进行，请输入任一个 11 位的素数……

作者不详

网络资源 开发可访问软件的指导原则可以在 <http://www-03.ibm.com/able/guidelines/software/access-software.html> 找到。

软件工具 用户界面开发

[目标] 用户界面开发工具使得软件工程师只需做有限的定制开发就可以建立复杂的

图形用户界面。这些工具提供了对可复用构件的访问，并且通过选择工具上预定义

的功能就可以建立用户界面。

[机制] 现代用户界面由一组可复用的构件组成, 这些构件与一些提供特殊特性的定制构件相结合。大多数用户界面的开发工具能够通过使用“拖放”功能来完成界面的设计。换句话说, 开发人员选择预定义的功能(例如, 表格构造器、交互机制、命令处理), 并将这些功能放置在所创建界面的环境中。

[代表性工具]^①

- LegaSuite GUI。由 Seagull Software (<http://www-304.ibm.com/partnerworld/gsd/solutiondetails.do?solution=1020&expand=true&lc=en>) 开发, 能够创建基于浏览器的

的图形用户界面(GUI)并且提供了对过时界面的再造功能。

- Motif Common Desktop Environment。由 Open Group (www.osf.org/tech/desktop/cde/) 开发, 是一个集成的图形用户界面, 用于开放系统桌面计算。它对数据、文件(图形化桌面)和应用系统的管理提供了单一的、标准的图形化界面。
- Alita Design 8.0。由 Altia (www.altia.com) 开发, 是一种可以在多种平台(例如, 自动的、手持的、工业的)上创建图形用户界面(GUI)的工具。

14.5 设计评估

一旦建立好可操作的用户界面原型, 必须对其进行评估, 以确定满足用户的需求。评估可以从非正式的“测试驱动”(比如用户可以临时提供一些反馈)到正式的设计研究(比如向一定数量的最终用户发放评估问题表, 采用统计学的方法进行评估)。

用户界面评估的循环如图 14-4 所示。完成设计模型后就开始建立第一级原型; 用户对该原型进行评估^②, 直接向设计者提供有关界面功效的建议, 如采用正式的评估技术(比如使用提问单、分级评分表), 这样设计者就能从调查结果中得到需要的信息(比如 80% 的用户不喜欢其中保存数据文件的机制); 针对用户的意见对设计进行修改, 完成下一级原型。评估过程不断进行下去, 直到不需要再修改为止。

原型开发方法是有效的, 但是否可以在建立原型以前就对用户界面的质量进行评估呢^③? 如果能够及早地发现和改正潜在的问题, 就可以减少评估循环执行的次数, 从而缩短开发时间。界面设计模型完成以后, 就可以运用下面的一系列评估标准 [Mor81] 对设计进行早期评审:

1. 系统及其界面的需求模型或书面规格说明的长度和复杂性在一定程度上体现了用户学习系统的难度。
2. 指定用户任务的个数以及每个任务动作的平均数在一定程度上体现了系统的交互时间

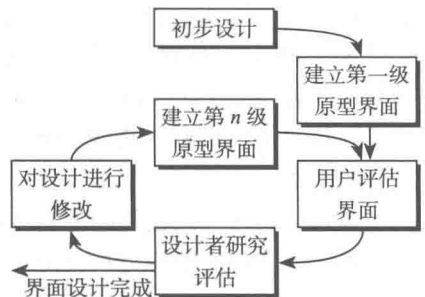


图 14-4 界面设计评估循环

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具

② 注意, 人类工程学和界面设计方面的专家也可对界面进行审查。这些审查叫作启发评估或认知走查。

③ 一些软件工程师更倾向于开发一个简单描画设计的用户界面模型, 称之为纸上原型, 使相关人员在交付任何程序资源之前先测试验证 UI 的内容。具体过程参见 http://www.paperprototyping.com/what_examples.html。

和系统的总体效率。

3. 设计模型中动作、任务和系统状态的数量体现了用户学习系统时所要记忆内容的多少。
4. 界面风格、帮助设施和错误处理协议在一定程度上体现了界面的复杂度和用户的接受程度。

一旦第一个原型完成以后,设计者就可以收集到一些定性和定量的数据以帮助进行界面评估。为了收集定性的数据,可以进行问卷调查,使用户能够评估界面原型。如果需要得到定量数据,就必须进行某种形式的定期研究分析。观察用户与界面的交互,记录以下数据:在标准时间间隔内正确完成任务的数量、使用动作的频度、动作顺序、观看屏幕的时间、出错的数目、错误的类型、错误恢复时间、使用帮助的时间、标准时间段内查看帮助的次数。这些数据可以用于指导界面修改。

有关用户界面评估方法的详细论述已超出了本书的范围,有兴趣的读者可以参考[Hac98]和[Sto05]等文献。

习题与思考题

14.1 描述一下你操作过的最好和最差的系统界面,采用本章介绍的相关概念对其进行评价。

14.2 在 14.1.1 节的基础上,再给出两条“把控制权交给用户”的设计原则。

14.3 在 14.1.2 节的基础上,再给出两条“减轻用户的记忆负担”的设计原则。

14.4 在 14.1.3 节的基础上,再给出两条“保持界面一致”的设计原则。

14.5 考虑下面几个交互应用(或者导师布置的应用):

- a. 桌面发布系统。
- b. 计算机辅助设计系统。
- c. 室内设计系统(如 14.3.2 节所描述的)。
- d. 大学课程自动注册系统。
- e. 图书管理系统。
- f. 基于网络的公共选举投票系统。
- g. 家庭银行系统。
- h. 导师布置的交互应用。

对上面给出的每个系统,开发用户模型、设计模型、心理模型和实现模型。

14.6 选择习题 14.5 中所列的任何一个系统,使用细化或面向对象的方法进行详细任务分析。

14.7 在 14.3.3 节提供的内容分析列表中至少再添加 5 个问题。

14.8 继续做习题 14.5,为你所选择的应用定义界面对象和动作。确定每个对象类型。

14.9 对于在习题 14.5 中所选的系统,开发一组带有主菜单和子菜单项定义的屏幕布局。

14.10 针对 SafeHome 系统,开发一组带有主菜单和子菜单项的屏幕布局,可以选择一种不同于图 14-3 的方法。

14.11 对于在习题 14.5、习题 14.7 和习题 14.8 中所完成的任务分析设计模型和分析任务,描述你采用的用户帮助设施。

14.12 举例说明为什么反应时间变动是一个问题。

14.13 开发一种能自动集成错误消息和用户帮助设施的方法。即系统能自动识别错误类型,并提供帮助窗口,给出改正错误的建议。进行合理且完整的软件设计,其中要考虑到合适的的数据结构和算法。

14.14 开发一个界面评估提问单,其中包括20个适用于大多数界面的通用问题。由10名同学完成你们所有人使用的交互系统的提问单。汇总你们的结果,并在班上做介绍。

扩展阅读与信息资源

尽管 Donald Norman 的著作 (《The Design of Everyday Things》, reissue edition, Basic Books, 2002) 不是专门阐述人机界面的,但其中涉及了进行有效设计的心理学,可以应用于用户界面的设计。对于那些非常关心高质量用户界面设计的人员,我们推荐此读物。Weinschenk 的著作 (《100 Things Every Designer Should Know About People》, New Riders, 2011) 并不特别侧重于软件,而是富有洞察力地提出了以用户为中心的设计。Johnson 的著作 (《Designing with the Mind in Mind》, Morgan Kaufman, 2010) 利用认知心理学开发有效的界面设计规则。

图形用户界面在现代计算世界中是无处不在的,无论是在 ATM、移动电话、汽车电子仪表盘、Web 站点,或者是在商业应用中,用户界面都为软件提供了窗口。正因如此,关于界面设计的书籍有很多,其中值得借鉴的有:

Ballard, 《Designing the Mobile User Experience》, Wiley, 2007。

Butow, 《User Interface Design for Mere Mortals》, Addison-Wesley, 2007。

Cooper 和他的同事, 《About Face 3: The Essentials of Interaction Design》, 3rd ed., Wiley, 2007。

Galitz, 《The Essential Guide to User Interface Design》 3rd ed., Wiley, 2007。

Goodwin 和 Cooper, 《Designing for the Digital Age: How to Create Human-Centered Products and Services》, Wiley, 2009。

Hartson 和 Pyla, 《The UX Book: Process and Guidelines For Ensuring a Quality User Experience》, Morgan Kaufman, 2012。

Lehikonen 和他的同事, 《Personal Content Experience : Managing Digital Life in the Mobile Age》, Wiley-Interscience, 2007。

Nielsen, 《Coordinating User Interfaces for Consistency》, Morgan-Kaufmann, 2006。

Pratt 和 Nunes, 《Interactive Design》, Rockport, 2013。

Rogers 和他的同事, 《Interactive Design: Beyond Human-Computer Interaction》, 3rd ed., Wiley, 2011。

Shneiderman 和他的同事, 《Designing the User Interface : Strategies for Effective Human-Computer Interaction》, 5th ed., Addison-Wesley, 2009。

Tidwell, 《Designing Interfaces》, O'Reilly Media, 2nd ed., 2011。

Johnson 的著作 (《GUI Bloopers: Common User Interface Design Don'ts and Do's》, 2nd ed., Morgan Kaufmann, 2007 和 《GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers》, Morgan Kaufmann, 2000) 对那些通过检查反例来实现高效学习的人提供了有用的指导。Cooper 编写的广受欢迎的书 (《The Inmates Are Running the Asylum》, Sams Publishing, 2004) 讨论了为什么高技术产品常令人感到疯狂,以及如何设计不让人疯狂的产品。

在网上有大量关于用户界面设计的信息,有关用户界面设计的最新参考文献列表可在 SEPA Web 站点 www.mhhe.com/pressman 找到。

质量管理

本书的这一部分将学习用来管理和控制软件质量的原理、概念和技术。在后面几章中会涉及下列问题：

- 高质量软件的一般特征是什么？
- 什么是软件质量保证？
- 软件测试需要应用什么策略？
- 使用什么方法才能设计出有效的测试用例？
- 有没有确保软件正确性的可行方法？
- 如何管理和控制软件开发过程中经常发生的变更？
- 使用什么标准和尺度评估需求模型、设计模型、源代码以及测试用例的质量？

回答了这些问题，就为保证生产出高质量软件做好了准备。

质量概念

要点浏览

概念: 究竟什么是软件质量? 答案不是想象中的那样容易, 当你看到它时你知道质量是什么, 可是, 它却不可捉摸, 难以定义。但是对于计算机软件, 质量是必须定义的, 这正是本章要讲的。

人员: 软件过程所涉及的每个人(软件工程师、经理和所有利益相关者)都对质量负有责任。

重要性: 你可以把事情做好, 或者再做一遍。如果软件团队在所有软件工程活动中强调质量, 就可以减少很多必需的返工, 结果是降低了成本, 更为重要的是

缩短了上市时间。

步骤: 为实现高质量软件, 必须具备四项活动: 已验证的软件工程过程和实践, 扎实的项目管理, 全面的质量控制, 具有质量保证基础设施。

工作产品: 满足客户需要的、准确而可靠地运行的、为所有使用者提供价值的软件。

质量保证措施: 通过检查所有质量控制活动的结果来跟踪质量, 通过在交付前检查错误, 在发布到现场后检查缺陷来衡量质量。

随着软件日益融入人们生活的方方面面, 提高软件质量的鼓声真的要擂响了。截至 20 世纪 90 年代, 大公司认识到由于软件达不到承诺的特性和功能, 每年浪费的钱财多达数十亿美元。更严重的是, 政府和产业界开始日益担心严重的软件缺陷有可能使重要的基础设施陷入瘫痪, 从而使花费超过数百亿美元。世纪之交, CIO 杂志一篇标题为“停止每年浪费 780 亿美元”的文章, 对“美国企业在不能如预期那样工作的软件上花费数十亿美元”[Lev01] 这一事实表示遗憾。InformationWeek[Ric01] 也表达了同样的担忧:

市场研究公司的 Standish Group 谈到, 尽管意愿良好, 但有缺陷的代码仍然是软件工业的幽灵, 计算机系统的故障时间高达 45%, 美国公司去年在丧失的生产率和修补上大约花费了一千亿美元, 这还不包括失去那些怒气冲冲的客户的代价。因为 IT 企业依赖基础软件包来开发应用, 所以糟糕代码也能损毁定制的应用。

多差的软件才是劣质软件呢? 人们对此的定义是不同的, 但是专家认为, 只要每 1000 行代码有 3 或 4 处缺陷就能使程序执行得很差, 大多数程序员每写 10 行代码大约注入一个错误, 许多商业产品有数百万行代码, 软件经销商至少将开发预算的一半花费在了测试时修

关键概念

- 质量的成本
- 足够好
- 责任
- 管理活动
- 质量
- 质量困境
- 质量维度
- 质量因素
- 量化观点
- 风险
- 安全

改错误上。

2005年, ComputerWorld[Hil05] 遗憾地表示, 劣质软件惹恼了几乎所有使用计算机的组织机构, 在计算机发生故障期间造成了工作时间损失、数据丢失或毁坏、销售时机丧失、IT支持与维护费用高昂, 以及客户满意度低等后果。一年后, InfoWorld[Fos06] 以“软件质量的可悲状况”作为主题书写了报告, 报告称质量问题依然没有任何改观。随着人们对软件质量的日益重视, 一项就 100000 位白领专业人士的调查 [Rog12] 显示软件质量工程师是“全美最幸福的工作者”!

现如今, 软件质量仍然是个问题, 但是应该责备谁? 客户责备开发人员, 认为草率的做法导致低质量的软件。开发人员责备客户 (和其他项目利益相关者), 认为不合理的交工日期以及连续不断的变更使开发人员在还没有完全验证时就交付了软件。谁说的对? 都对, 这正是问题所在。本章把软件质量作为一个概念, 考查在软件工程实践中为什么软件质量值得认真考虑。

15.1 什么是质量

Robert Persig[Per74] 在他的神秘之书《Zen and the Art of Motorcycle Maintenance》中就我们称为“质量”的东西发表了看法:

质量……你知道它是什么, 也不知道它是什么。这样说是自相矛盾的, 但没有比这更好的说法了——这样说更有质量。但是当要试图说明质量是什么时, 除了我们知道的这些之外, 总是所知了了! 没有什么好谈论的, 但是如果不能说明质量是什么, 又怎能知道质量是什么, 或者又怎能知道质量甚至是否存在呢? 如果没有人知道质量是什么, 那么所有实际用途就根本不存在, 但是所有实际用途确实是存在的。质量等级依据其他哪些东西来划分? 为什么人们将机会给某些东西而把其他东西扔到垃圾堆? 显而易见某些东西好于其他东西……但是, 好在什么地方呢? ……所以就任凭金属的轮子一圈又一圈地转动, 而找不到摩擦力在哪里。质量到底是什么? 什么是质量?

的确, 什么是质量?

在更为实用的层面上, 哈佛商学院的 David Garvin[Gar84] 给出了建议: “质量是一个复杂多面的概念”, 可以从 5 个不同的观点来描述。先验论观点 (如 Persig) 认为质量是马上就能识别的东西, 却不能清楚地定义。用户观点是从最终用户的具体目标来考虑的。如果产品达到这些目标, 就是有质量的。制造商观点是从产品原始规格说明的角度来定义质量, 如果产品符合规格说明, 就是有质量的。产品观点认为质量是产品的固有属性 (比如功能和特性)。最后, 基于价值的观点根据客户愿意为产品支付多少钱来评测质量。实际上, 质量涵盖所有这些观点, 或者更多。

设计质量是指设计师赋予产品的特性。材料等级、公差和性能等规格说明决定了设计质量。如果产品是按照规格说明书制造的, 那么使用较高等级的材料, 规定更严格的公差和更高级别的性能, 产品的设计质量就能提高。

在软件开发中, 设计质量包括设计满足需求模型规定的功能和特性的程度。符合质量关注的是实现遵从设计的程度以及所得到的系统满足需求和性能目标的程度。

但是, 设计质量和符合质量是软件工程师必须考虑的唯一问题吗?

提问 在看待质量方面有哪些不同的方法?

引述 人们记不住你做一项工作有多快——但总能记住你做得有多好。

Howard Newton

Robert Glass [Gla98] 认为它们之间比较“直观的”关系符合下面的公式：

用户满意度 = 合格的产品 + 好的质量 + 按预算和进度安排交付

总之，Glass 认为质量是重要的。但是，如果用户不满意，其他任何事情也就都不重要了。DeMarco [DeM98] 同意这个观点，他认为：“产品的质量是一个函数，该函数确定了它在多大程度上使这个世界变得更好。”这个质量观点的意思就是：如果一个软件产品能给最终用户带来实质性的益处，那么他们可能会心甘情愿地忍受偶尔的可靠性或性能问题。

15.2 软件质量

高质量的软件是一个重要目标，即使最疲倦的软件开发人员也会同意这一点。但是，如何定义软件质量呢？在最一般的意义上，软件质量可以这样定义：在一定程度上应用有效的软件过程，创造有用的产品，为生产者和使用者提供明显的价值。^①

毫无疑问，对上述定义可以进行修改、扩展以及无休止的讨论。针对本书的论题来说，该定义强调了以下三个重要的方面：

1. 有效的软件过程为生产高质量的软件产品奠定了基础。过程的管理方面所做的工作是检验和平衡，以避免项目混乱（低质量的关键因素）。软件工程实践允许开发人员分析问题、设计可靠的解决方案，这些都是生产高质量软件的关键所在。最后，诸如变更管理和技术评审等普适性活动与其他部分的软件工程活动密切相关。
2. 有用的产品是指交付最终用户要求的内容、功能和特征，但最重要的是，以可靠、无误的方式交付这些东西。有用的产品总是满足利益相关者明确提出的那些需求，另外，也要满足一些高质量软件应有的隐性需求（例如易用性）。
3. 通过为软件产品的生产者和使用者增值，高质量软件为软件组织和最终用户群体带来了收益。软件组织获益是因为高质量的软件在维护、改错及客户支持方面的工作量都降低了，从而使软件工程师减少了返工，将更多的时间花费在开发新的应用上，软件组织因此而获得增值。用户群体也得到增值，因为应用所提供的有用的能力在某种程度上加快了一些业务流程。最后的结果是：（1）软件产品的收入增加；（2）当应用可支持业务流程时，收益更好；（3）提高了信息可获得性，这对商业来讲是至关重要的。

提问 如何为软件质量下一个最好的定义？

引述 作为卓越的代名词，一些人并不能适应需要杰出素质的环境。

Steve Jobs

15.2.1 Garvin 的质量维度

David Garvin[Gar87] 建议采取多维的观点考虑质量，包括从符合性评估到抽象的（美学）观点。尽管 Garvin 的 8 个质量维度没有专门为软件制定，但考虑软件质量时依然可以使用。

性能质量。软件是否交付了所有的内容、功能和特性？这些内容、功能和特性在某种程度上是需求模型所规定的一部分，可以为最终用户提供价值。

特性质量。软件是否提供了使第一次使用的最终用户感到惊喜的特性？

可靠性。软件是否无误地提供了所有的特性和能力，当需要使用该软件时，它是否是可

建议 当应用采用了 Garvin 的质量维度时，可以使用雷达图提供每个 Garvin 质量维度的可视化表现形式。

① 该定义节选自 [Bes04]，取代该书前几版中出现的更面向生产的观点。

用的，是否无错地提供了功能？

符合性。软件是否遵从本地的和外部的与应用领域相关的软件标准，是否遵循了事实存在的设计惯例和编码惯例？例如，对于菜单选择和数据输入等用户界面的设计是否符合人们已接受的设计规则？

耐久性。是否能够对软件进行维护（变更）或改正（改错），而不会粗心大意地产生意想不到的副作用？随着时间的推移，变更会使错误率或可靠性变得更糟吗？

适用性。软件能在可接受的短时期内完成维护（变更）和改正（改错）吗？技术支持人员能得到所需的所有信息以进行变更和修正缺陷吗？Douglas Adams[Ada93]挖苦地评论道：“可能发生故障的东西与不可能发生故障的东西之间的差别是，当前者发生了故障时，通常是不可能发现和补救的”。

审美。毫无疑问，关于什么是美的，我们每个人有着不同的、非常主观的看法。可是，我们中的大多数都同意美的东西具有某种优雅、特有的流畅和醒目的外在，这些都是很难量化的，但显然是不可缺少的，美的软件具有这些特征。

感知。在某些情况下，一些偏见将影响人们对质量的感知。例如，有人给你介绍了一款软件产品，该软件产品是由过去曾经生产过低质产品的厂家生产的，你的自我保护意识将会增加，你对于当前软件产品质量的感知力可能受到负面影响。类似地，如果厂家有极好的声誉，你将能感觉到好的质量，甚至在实际质量并不真的如此时，你也会这样想。

Garvin 的质量维度提供了对软件质量的“软”评判。这些维度中的多数（不是所有）只能主观地考虑。正因如此，也需要一套“硬”的质量因素，这些因素可以宽泛地分成两组：（1）可以直接测量的因素（例如，测试时发现的缺陷数）；（2）只能间接测量的因素（例如，可用性或可维护性）。在任何情况下，必须进行测量，应把软件和一些基准数据进行比较来确定质量。

15.2.2 McCall 的质量因素

McCall、Richards 和 Walters[McC77]提出了影响软件质量因素的一种有用的分类。这些软件质量因素侧重于软件产品的三个重要方面：操作特性、承受变更的能力以及对新环境的适应能力，如图 15-1 所示。

针对图 15-1 中所提到的因素，McCall 及他的同事提供了如下描述：

正确性。程序满足其需求规格说明和完成用户任务目标的程度。

可靠性。期望程序以所要求的精度完成其预期功能的程度。需要提醒大家注意的是，还有更完整的可靠性定义（第 16 章）。

效率。程序完成其功能所需的计算资源和代码的数量。

完整性。对未授权的人员访问软件或数据的可控程度。

易用性。对程序进行学习、操作、准备输入和解释输出所需要的工作量。

可维护性。查出和修复程序中的一个错误所需要的工作量。（这是一个

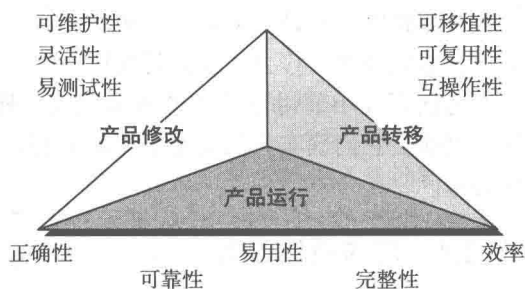


图 15-1 McCall 的软件质量因素

引述 满足进度计划的喜悦已经被遗忘很久之后，低质量所带来的痛苦依然挥之不去。

Karl Weigers

非常受限的定义。)

灵活性。修改一个运行的程序所需的工作量。

易测试性。测试程序以确保它能完成预期功能所需要的工作量。

可移植性。将程序从一个硬件和软件系统环境移植到另一个环境所需要的工作量。

可复用性。程序(或程序的一部分)可以在另一个应用中使用的程度。这与程序所执行功能的封装和范围有关。

互操作性。将一个系统连接到另一系统所需要的工作量。

要想求得这些质量因素的直接测度^①是困难的,且在有些情况下是不可能的。事实上,由 McCall 等人定义的度量仅能间接地测量。不过,使用这些因素评估应用的质量可以真实地反映软件的质量。

15.2.3 ISO 9126 质量因素

ISO 9126 国际标准的制定是试图标识计算机软件的质量属性。这个标准标识了 6 个关键的质量属性。

功能性。软件满足已确定要求的程度,由以下子属性表征:适合性、准确性、互操作性、依从性和安全性。

可靠性。软件可用的时间长度,由以下子属性表征:成熟性、容错性和易恢复性。

易用性。软件容易使用的程度,由以下子属性表征:易理解性、易学习性和易操作性。

效率。软件优化使用系统资源的程度,由以下子属性表征:时间特性和资源利用特性。

可维护性。软件易于修复的程度,由以下子属性表征:易分析性、易改变性、稳定性和易测试性。

可移植性。软件可以从一个环境移植到另一个环境的容易程度,由以下子属性表征:适应性、易安装性、符合性和易替换性。

与前面几小节讨论的软件质量因素一样,ISO 9126 中的质量因素不一定有助于直接测量。然而,它们确实为间接测量提供了有价值的基础,并为评估系统质量提供了一个优秀的检查单。

15.2.4 定向质量因素

15.2.1 节和 15.2.2 节提出的质量维度和因素关注软件的整体,可以用其作为表征应用质量的一般性指标。软件团队可以提出一套质量特征和相关的问题以调查软件满足每个质量因素的程度。例如:McCall 把易用性看作重要的质量因素。当要求评审用户界面和评估易用性时,该如何进行?可能要从 McCall 提出的子属性——易理解性、易学习性和易操作性开始,但是在实用的意义上,这些子属性表示什么意思呢?

为了进行评价,需要说清楚界面具体的、可测量的(或至少是可识别的)属性。例如下

建议 尽管为这里提到的质量因素制定量化测量是很诱人的,但你也可以创建一个简单的属性清单来提供显示质量因素的可靠指标。

引述 当从业人员想将事情做正确或做得更好时,任何活动都将变得富有创造性。

John Updike

① 直接测度意味着存在一个简单可计算的值。该值为被考察的属性提供直接的指标。例如,程序的“规模”可以通过计算代码的行数直接测量。

面这些属性 [Bro03]。

直觉。界面遵照预期使用模式的程度，使得即使是新手也能不经过专门培训而开始使用。

- 界面布局易于理解吗？
- 界面操作容易找到和上手吗？
- 界面使用了可识别的隐喻吗？
- 输入的安排可尽量少敲击键盘和点击鼠标吗？
- 界面符合三个重要原则吗（第14章）？
- 美学的运用有助于理解和使用吗？

效率。定位或初步了解操作和信息的程度。

- 界面的布局和风格可以使用户有效地找到操作和信息吗？
- 一连串的操作（或数据输入）可以用简单动作实现吗？
- 输出的数据和显示的内容是否能立即被理解？
- 分层操作是否组织得能使用户完成某项工作所需导航的深度最小？

健壮性。软件处理有错的输入数据或不恰当的用户交互的程度。

- 如果输入了规定边界上的数据或恰好在边界外的数据，软件能识别出错误吗？更为重要的是，软件还能继续运行而不出错或性能不下降吗？
- 界面能识别出常见的可识别错误或操作错误，并能清晰地指导用户回到正确的轨道上来吗？
- 若发现了错误的情况（与软件功能有关），界面是否提供了有用的诊断和指导？

丰富性。界面提供丰富特征集的程度。

- 界面是否能按照用户的特定要求进行定制？
- 界面是否提供宏操作以使用户将单个的行为或命令当作一连串的常用操作？

当界面设计展开后，软件团队将评审设计原型，询问他们所关注的问题。如果对这些问题的大多数回答是“肯定的”，用户界面就具备了高质量。应该为每个待评估的质量因素开发出类似的一组问题。

15.2.5 过渡到量化观点

前面几节讨论了一组测量软件质量的定性因素。软件界也力图开发软件质量的精确的测量，但有时又会为活动的主观性而受挫。Cavano 和 McCall[Cav78] 讨论了这种情形：

质量评定是日常事件（葡萄酒品尝比赛、运动赛事（如体操）、智力竞赛等）中的一个关键因素。在这些情况下，质量是以最基本、最直接的方式来判断的：在相同的条件和预先决定的概念下将对象进行并列对比。葡萄酒的质量可以根据清澈度、颜色、酒花和味道等来判断。然而，这种类型的判断是很主观的，最终的结果必须由专家给出。

主观性和特殊性也适用于软件质量的评定。为了帮助解决这个问题，需要对软件质量有一个更精确的定义，同样，为了客观分析，需要产生软件质量的定量测量方法……既然没有这种事物的绝对知识，就不要期望精确测量软件质量，因为每一种测量都是部分地不完美的。Jacob Bronkowski 这样描述知识的自相矛盾现象：“年复一年，我们设计更精准

建议 当面临质量困境时（每个人都会在某个时期面对它），尽力取得平衡——用足够的工作量去开发质量可接受的产品，而不是将项目葬送掉。

的仪器用以更精细地观察自然界，可是当我们留心这些观察数据，却很不愉快地发现这些数据依然模糊时，我们感到它们还和过去一样不确定。”

软件度量可应用于软件质量的定量评估。在所有的情况下，这些度量都表示间接的测量。也就是说，我们从不真正测量质量，而是测量质量的一些表现。复杂因素在于所测量的变量和软件质量间的精确关系。

15.3 软件质量困境

在网上发布的一篇访谈 [Ven03] 中，Bertrand Meyer 这样论述我所称谓的质量困境：

如果生产了一个存在严重质量问题的软件系统，你将受到损失，因为没有人想去购买。另一方面，如果你花费无限的时间、极大的工作量和高额的资金来开发一个绝对完美的软件，那么完成该软件将花费很长的时间，生产成本是极其高昂的，甚至会破产。要么错过了市场机会，要么几乎耗尽所有的资源。所以企业界的人努力达到奇妙的中间状态：一方面，产品要足够好，不会立即被抛弃（比如在评估期）；另一方面，又不是那么完美，不需花费太长时间和太多成本。

软件工程师应该努力生产高质量的系统，在这一过程中如果能采用有效的方法就更好了。但是，Meyer 所讨论的情况是现实的，甚至对于最好的软件工程组织，这种情况也表明了一种两难的困境。

建议 虽然开发这里提到的质量因素的量化测量是临时的，但还是可以创建一个简单的属性检查表，这些属性可以提供表征质量因素的可靠指示。

15.3.1 “足够好”的软件

坦率地说，如果我们准备接受 Meyer 的观点，那么生产“足够好”软件是可接受的吗？对于这个问题的答案只能是“肯定的”，因为大型软件公司每天都在这么做。这些大公司生产带有已知缺陷的软件，并发布给大量的最终用户。他们认识到，1.0 版提供的一些功能和特性达不到最高质量，并计划在 2.0 版改进。他们这样做时，知道有些客户会抱怨，但他们认识到上市时间胜过更好的质量，只要交付的产品“足够好”。

到底什么是“足够好”？足够好的软件提供用户期望的高质量功能和特性，但同时也提供了其他更多的包含已知错误的难解的或特殊的功能和特性。软件供应商希望广大的最终用户忽视错误，因为他们对其他的应用功能是如此满意。

这种想法可能引起许多读者的共鸣。如果你是其中之一，我只能请你考虑一些论据来反对“足够好”。

诚然，“足够好”可能在某些应用领域和几个主要的软件公司起作用。毕竟，如果一家公司有庞大的营销预算，并能够说服足够多的人购买 1.0 版本，那么该公司已经成功地锁定了这些用户。正如前面所指出的，可以认为，公司将在以后的版本提高产品质量。通过提供足够好的 1.0 版，公司垄断了市场。

如果你所在的是一个公司，就要警惕这一观念，当你交付一个足够好的（有缺陷的）产品时，你是冒着永久损害公司声誉的风险。你可能再也没有机会提供 2.0 版本了，因为异口同声的不良评论可能会导致销售暴跌乃至公司关门。

如果你工作在某个应用领域（如实时嵌入式软件），或者你构建的是与硬件集成的应用软件（如汽车软件、电信软件），那么一旦交付了带有已知错误的软件（也许是一时疏忽），就可能使公司处于代价昂贵的诉讼之中。在某些情况下，甚至可能是刑事犯罪，没有人想要

足够好的飞机航空电子系统软件！

因此，如果你认为“足够好”是一个可以解决软件质量问题的捷径，那么要谨慎行事。“足够好”可以起作用，但只是对于少数几个公司，而且只是在有限的几个应用领域。^①

15.3.2 质量的成本

关于软件成本有这样的争论：我们知道，质量是重要的，但是花费时间和金钱——花费太多的时间和金钱才能达到我们实际需要的软件质量水平。表面上看，这种说法似乎是合理的（见本节前面 Meyer 的评论）。毫无疑问，质量好是有成本的，但质量差也有成本——不仅是对必须忍受缺陷软件的最终用户，而且是对已经开发且必须维护该软件的软件组织。真正的问题是：我们应该担心哪些成本？要回答这个问题，你必须既要了解实现质量的成本，又要了解低质量软件的成本。

质量成本包括追求质量过程中或在履行质量有关的活动中引起的费用以及质量不佳引起的下游费用等所有费用。为了解这些费用，一个组织必须收集度量数据，为目前的质量成本提供一个基准，找到降低这些成本的机会，并提供一个规范化的对比依据。质量成本可分为预防成本、评估成本和失效成本。

预防成本包括：（1）计划和协调所有质量控制和质量保证所需管理活动的成本；（2）为开发完整的需求模型和设计模型所增加的技术活动的成本；（3）测试计划的成本；（4）与这些活动有关的所有培训成本。

评估成本包括为深入了解产品“第一次通过”每个过程的条件而进行的活动。评估成本的例子包括：（1）对软件工程工作产品进行技术评审的成本；（2）数据收集和度量估算的成本；（3）测试和调试（第 17～19 章）的成本。

失效成本是那些在将产品交付客户之前若没有出现错误就不会发生的费用。失效成本可分为内部失效成本和外部失效成本。内部失效成本发生在软件发布之前发现错误时，内部失效成本包括：（1）为纠正错误进行返工（修复）所需的成本；（2）返工时无意中产生副作用，必须对副作用加以缓解而发生的成本；（3）组织为评估失效的模型而收集质量数据，由此发生的相关成本。外部失效成本是在产品已经发布给客户之后发现了缺陷时的相关成本。外部成本的例子包括：解决投诉，产品退货和更换，帮助作业支持，以及与保修工作相关的人力成本。不良的声誉和由此产生的业务损失是另一个外部失效成本，这是很难量化但非常现实的。生产了低质量的软件产品时，不好的事情就要发生。

在对拒绝考虑外部失效成本的软件开发者的控告书中，Cem Kaner [Kan95] 是这样说的：

许多外部失效成本（如声誉的损失）都难以量化，因此许多企业在计算其成本效益的权衡时忽视了这些成本。还有一些外部失效成本可以降低（例如，通过提供更便宜、质量更低的产品，售后支持，或向客户收取支持费用），这些都不会增加用户的满意度。质量工程师靠忽视使用不良产品的客户成本，鼓励做出相关的质量决策，这种决策只会欺骗客户，而不会使客户高兴。

正如预料的那样，当我们从预防到检查内部失效成本和外部失效成本时，找到并修复

建议 不要担心预防成本显著增加。放心，你的投资将会获得良好的回报。

引述 把事情做对比解释为什么做错花费的时间更少。

H. W. Longfellow

① 关于“足够好”软件的优缺点的有价值的探讨请参阅 [Bre02]。

错误或缺陷的相关成本会急剧增加。根据 Boehm 和 Basili 收集的数据 [Boe01b] 以及 Cigital Inc[Cig07] 的阐述, 图 15-2 说明了这一现象。

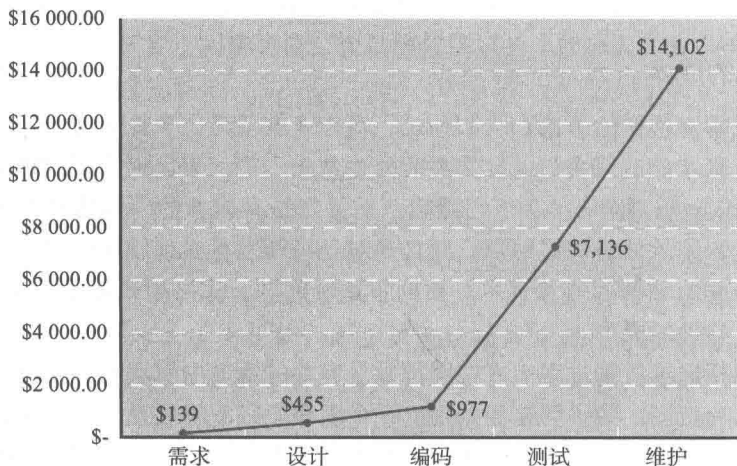


图 15-2 改正错误和缺陷的相对成本 [Boe01b]

在代码生成期纠正缺陷的行业平均成本是每个错误大约 977 美元, 而在系统测试期, 纠正同样错误的行业平均成本是每个错误 7136 美元。[Cig07] 认为, 一个大型应用程序在编码期会引入 200 个错误。

根据行业平均水平的数据, 在编码阶段发现和纠正缺陷的成本是每个缺陷 977 美元, 因此, 在本阶段纠正这 200 个“关键”缺陷的总费用大约是 195400 美元 (200×977 美元)。

行业平均水平数据显示, 在系统测试阶段发现和纠正缺陷的代价是每个缺陷 7136 美元。在这种情况下, 假定该系统测试阶段发现大约 50 个关键缺陷 (或者 Cigital 在编码阶段只发现了这些缺陷的 25%), 发现和解决这些缺陷的代价将是大约 356800 美元 (50×7136 美元)。这将导致 150 处关键错误未被发现和矫正。在维护阶段发现和解决这些遗留的 150 个缺陷的代价将是 2115300 美元 (150×14102 美元)。因此, 在编码阶段之后发现和解决这 200 个缺陷的代价将是 2472100 美元 (2115300 美元 + 356800 美元)。

即使软件组织所花费的是行业平均水平的一半 (大多数企业尚不知道他们的成本!), 与早期的质量控制和保证活动 (进行需求分析和设计) 有关的成本节约也是不得不做的。

SafeHome 质量问题

[场景] Doug Miller 的办公室, SafeHome 软件项目开始。

[人物] Doug Miller (SafeHome 软件工程团队经理) 和产品软件工程团队的其他成员。

[对话]

Doug: 我正在看一份关于软件缺陷修改成本的行业报告, 缺陷修改成本令人警醒。

Jamie: 我们已经准备好为每一个功能需求

开发测试用例。

Doug: 好的, 我注意到修复在测试期间发现的一处缺陷要比在编码期间发现并修复一处缺陷花费八倍的工作量。

Vinod: 我们正使用结对编程法, 所以我们应能捕获编码期间的大多数缺陷。

Doug: 我认为你没有抓住重点, 软件质量不只是简单地去除编码错误。我们需要关注项目质量目标, 保证不断变更的软件产

品满足这些目标。

Jamie: 您的意思是可用性、安全性和可靠性那些问题吗?

Doug: 是的,我们需要在软件过程中加入检查机制,以监视过程是朝着质量目标方向的。

Vinod: 难道我们不能在完成第一个原型之

后进行质量检查吗?

Doug: 恐怕不能,我们必须在项目早期建立质量文化。

Vinod: 您要我们做什么,Doug?

Doug: 我认为需要寻找一种使我们能监视SafeHome产品质量的技术,让我们考虑一下,明天再讨论这个话题。

15.3.3 风险

本书的第1章写道:“人们拿自己的工作、自己的舒适、自己的安全、自己的娱乐、自己的决定以及自己的生命在计算机软件上下赌注。最好这是正确的。”这意味着,低质量的软件为开发商和最终用户都增加了风险。前面讨论了这些风险(成本)中的一个,但设计和实现低劣的应用所带来的损失并不总是限于美元和时间,一个极端的例子[Gag04]可能有助于说明这一点。

整个2000年11月份,在巴拿马的一家医院,28位病人在治疗多种癌症的过程中受到了过量的伽马射线照射。此后数月内,其中5例死于辐射病,15人发展成严重的并发症。是什么造成了这一悲剧?这是因为医院的技术人员对一家美国公司开发的软件包进行了修改,以计算每位病人的辐射剂量的变更值。

为了获取额外的软件功能,三位巴拿马医疗物理学家“调整”了软件,他们被指控犯有二级谋杀罪。同时,这家美国软件公司正在两个国家面临着严重的诉讼。Gage和McCormick评论道:

这不是一个讲给医疗技术人员的警示故事,尽管由于误解或误用了技术,他们要为了免于牢狱之灾而据理力争。这也不是一个关于人类如何受到伤害的故事,或者更糟的是被设计不良或说明不清的软件伤害,尽管这样的例子比比皆是。这是任何一个计算机程序设计者都应当铭记的教训:软件质量问题很重要,不管软件是嵌入汽车引擎中、工厂里的机械手臂中,还是嵌入医院的治疗设备中,这些应用必须做到万无一失,低劣部署的代码可以杀人。

质量低劣导致风险,其中一些风险会非常严重。

15.3.4 疏忽和责任

这种情况太常见了。政府或企业雇用较大的软件开发商或咨询公司来分析需求,然后设计和创建一个基于软件的“系统”,用以支撑某个重大的活动。系统可能支持主要的企业功能(例如,养老金管理),或某项政府职能(例如,卫生保健管理或国土安全)。

工作始于双方良好的意愿,但是到了系统交付时,情况已变得糟糕,系统延期,未能提供预期的特性和功能,而且易出错,不能得到客户的认可,接下来就要打官司。

在大多数情况下,顾客称开发商马虎大意(已带到了软件实践中),因此拒绝付款。而开发商则常常声称,顾客一再改变其要求,并在其他方面破坏了开发伙伴关系。无论是哪一种情况,交付系统的质量都会有问题。

15.3.5 质量和安全

随着基于Web的系统和移动系统重要性的增加,应用的安全性已变得日益重要。简而

言之，没有表现出高质量的软件比较容易被攻击。因此，低质量的软件会间接地增加安全风险，随之而来的是费用和问题。

在 ComputerWorld 的一篇访谈中，作者和安全专家 Gary McGraw 这样评论 [Wil05]：

软件安全与质量息息相关。必须一开始就在设计、构建、测试、编码阶段以及在整个软件生命周期（过程）中考虑安全性、可靠性、可得性、可信性。即使是已认识到软件安全问题的人也会主要关注生命周期的晚些阶段。越早发现软件问题越好。有两种类型的软件问题，一种是隐藏的错误，这是实现的问题。另一种是软件缺陷，这是设计中的构建问题。人们对错误关注太多，却对缺陷关注不够。

要构造安全的系统，就必须注重质量，并必须在设计时开始关注。本书第二部分讨论的概念和方法可以导出减少“缺陷”的软件架构。我们将在第 20 章更详细地讨论软件安全工程。

15.3.6 管理活动的影响

软件质量受管理决策的影响往往和受技术决策的影响是一样的。即使最好的软件工程实践也能被糟糕的商业决策和有问题的项目管理活动破坏。

本书第四部分讨论软件过程环境下的项目管理。每个项目任务开始时，项目领导人都要作决策，这些决策可能对产品质量有重大影响。

估算决策。在确定交付日期和制定总预算之前，给软件团队提供项目估算数据是很少见的。反而是团队进行了“健康检查”，以确保交付日期和里程碑是合理的。在许多情况下，存在着巨大的上市时间压力，迫使软件团队接受不现实的交付日期。结果，由于抄了近路，可以获得更高质量软件的活动被忽略掉了，产品质量受到损害。如果交付日期是不合理的，那么坚持立场就是重要的。这就解释了为什么你需要更多的时间，或者也可以建议在指定的时间交付一个（高质量的）功能子集。

进度安排决策。一旦建立了软件项目时间表（第 25 章），就会按照依赖性安排任务的先后顺序。例如，由于 A 构件依赖于 B、C 和 D 构件中的处理，因此直到 B、C 和 D 构件完全测试后，才能安排 A 构件进行测试。项目计划将反映这一点。但是，如果时间很紧，为了做进一步的关键测试，A 必须是可用的。在这种情况下，可能会决定在没有其附属构件（这些附属构件的运行要稍落后于时间表）的情况下测试 A，这样对于交付前必须完成的其他测试，就可以使用 A 了，毕竟，期限正在逼近。因此，A 可能有隐藏的缺陷，只有晚些时候才能发现，质量会受到影响。

面向风险的决策。风险管理（第 26 章）是成功软件项目的关键特性之一。必须知道哪里可能会出问题，并建立一项如果确实出问题时的应急计划。太多的软件团队喜欢盲目乐观，在什么都不会出问题的假设下建立开发计划。更糟的是，他们没有办法处理真的出了差错的事情。结果，当风险变成现实后，便会一片混乱，并且随着疯狂程度的上升，质量水平必然下降。

用 Meskimen 定律能最好地概括软件质量面临的困境——从来没有时间做好，但总是有时间再做一遍。我的建议是，花点时间把事情做好，这几乎从来都不是错误的决定。

15.4 实现软件质量

良好的软件质量不会自己出现，它是良好的项目管理和扎实的软件工程实践的结果。帮

助软件团队实现高质量软件的四大管理和实践活动是：软件工程方法、项目管理技术、质量控制活动以及软件质量保证。

15.4.1 软件工程方法

如果希望建立高质量的软件，就必须理解要解决的问题。还须能够创建一个符合问题的设计，该设计同时还要具备一些性质，这些性质可以使我们得到具有 15.2 节讨论过的质量维度和因素的软件。

本书第二部分提出了一系列概念和方法，可帮助我们获得对问题合理完整的理解和综合性设计，从而为构建活动建立了坚实的基础。如果应用这些概念，并采取适当的分析和设计方法，那么创建高质量软件的可能性将大大提高。

15.4.2 项目管理技术

在 15.3.6 节已经讨论了不良管理决策对软件质量的影响。其中的含义是明确的：如果（1）项目经理使用估算以确认交付日期是可以达到的；（2）进度依赖关系是清楚的，团队能够抵抗走捷径的诱惑；（3）进行了风险规划，这样出了问题就不会引起混乱，软件质量将受到积极的影响。

此外，项目计划应该包括明确的质量管理和变更管理技术。导致良好项目管理实践的技术将在本书第四部分讨论。

提问 需要做些什么以对质量产生积极的影响？

15.4.3 质量控制

质量控制包括一套软件工程活动，以帮助确保每个工作产品符合其质量目标。评审模型以确保它们是完整的和一致的。检查代码，以便在测试开始前发现和纠正错误。应用一系列的测试步骤以发现逻辑处理、数据处理以及接口通信中的错误。当这些工作成果中的任何一个不符合质量目标时，测量和反馈的结合使用使软件团队可以调整软件过程。本书第三部分的其余章节对质量控制活动进行了详细的讨论。

提问 软件质量控制是什么？

15.4.4 质量保证

质量保证建立基础设施，以支持坚实的软件工程方法，合理的项目管理和质量控制活动——如果你打算建立高品质软件，那么所有这些都是关键活动。此外，质量保证还包含一组审核和报告功能，用以评估质量控制活动的有效性和完整性。质量保证的目标是为管理人员和技术人员提供所需的数据，以了解产品的质量状况，从而理解和确信实现产品质量的活动在起作用。当然，如果质量保证中提供的数据出现了问题，那么处理问题和使用必要的资源来解决质量问题是管理人员的职责。软件质量保证将在第 16 章详细论述。

网络资源 有关软件质量保证的有用资料可以在下列站点找到：
www.niwotridge.com/Resources/P-M-SWEResources/SoftwareQualityAssurance.htm

习题与思考题

15.1 描述在申请学校之前你将怎样评估一所大学的质量。哪些因素重要？哪些因素关键？

- 15.2 Garvin[Gar84] 描述了质量的 5 种不同的观点, 用一个或多个您熟悉的知名电子产品为每个观点举一个例子。
- 15.3 使用 15.2 节提出的软件质量定义, 不使用有效的过程来创建能提供明显价值的有用产品, 你认为可能吗? 解释你的答案。
- 15.4 为 15.2.1 节介绍的 Garvin 的每个质量维度添加两个额外的问题。
- 15.5 McCall 质量因素是在 20 世纪 70 年代提出的, 自提出以来, 几乎计算的每个方面都发生了翻天覆地的变化, 然而, McCall 质量因素继续适用于现代软件。基于这一事实你是否能得出一些结论?
- 15.6 使用 15.2.3 节所述 ISO9126 质量因素中“维护性”的子属性, 提出一组问题, 探讨是否存在这些属性, 仿效 15.2.4 节所示的例子。
- 15.7 用你自己的话描述软件质量困境。
- 15.8 什么是“足够好”的软件? 说出具体公司的名字, 以及你认为运用足够好思想开发的具体产品。
- 15.9 考虑质量成本的 4 个方面, 你认为哪个方面是最昂贵的, 为什么?
- 15.10 进行网络搜索, 寻找直接由低劣的软件质量所致“风险”的其他三个例子。考虑从 <http://catless.ncl.ac.uk/risks> 开始搜索。
- 15.11 质量和安全是一回事吗? 请加以解释。
- 15.12 解释为什么我们许多人仍在沿用 Meskimen 定律, 使得软件业如此的原因是什么?

扩展阅读与信息资源

Chemutri (《Master Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers》, Ross Publishing, 2010)、Henry 和 Hanlon (《Software Quality Assurance》, Prentice Hall, 2008)、Khan 和他的同事 (《Software Quality: Concepts and Practice》, Alpha Science International, Ltd., 2006)、O'Regan (《A Practical Approach to Software Quality》, Springer, 2002) 以及 Daughtrey (《Fundamental Concepts for the Software Quality Engineer》, ASQ Quality Press, 2001) 的书讨论了软件质量的基本概念。

Duvall 和他的同事 (《Continuous Integration: Improving Software Quality and Reducing Risk》, Addison-Wesley, 2007)、Tian (《Software Quality Engineering》, Wiley-IEEE Computer Society Press, 2005)、Kandt (《Software Engineering Quality Practices》, Auerbach, 2005)、Godbole (《Software Quality Assurance: Principles and Practice》, Alpha Science International, Ltd., 2004) 以及 Galin (《Software Quality Assurance: From Theory to Implementation》, Addison-Wesley, 2003) 介绍了软件质量保证的具体做法。Sterling (《Managing Software Debt》, Addison-Wesley, 2010) 以及 Stamelos 和 Sfetos (《Agile Software Development Quality Assurance》, IGI Global, 2007) 讨论了在敏捷过程环境中的质量保证问题。

可靠的设计可以实现软件的高质量。Jayasawal 和 Patton (《Design for Trustworthy Software》, Prentice Hall, 2006) 以及 Ploesch (《Contracts, Scenarios and Prototypes》, Springer, 2004) 讨论了开发“健壮”软件的工具和技术。

测量是软件质量工程的一个重要部分。Jones 和 Bonsignour (《The Economic of Software Quality》, Addison-Wesley, 2011)、Ejiogu (《Software Metrics: The Discipline of Software Quality》, BookSurge Publishing, 2005)、Kan (《Metrics and Models in Software Quality Engineering》, Addison-Wesley, 2002) 以及 Nance 和 Arthur (《Managing Software Quality》, Springer, 2002) 讨论了重要的质量相关的测度和模型。Evans (《Achieving Software Quality through Teamwork》, Artech House Publishers, 2004) 考虑了软件质量中团队方面的问题。

软件质量方面的大量信息可以在网上获得, 关于软件质量方面最新的参考文献可以在 SEPA 网站 www.mhhe.com/pressman 的“software engineering resources”下找到。

软件质量保证

要点浏览

概念：仅仅嘴上说“软件质量重要”是不够的，还必须：（1）明确给出你所说的“软件质量”的含义；（2）提出一组活动，这些活动将有助于保证每个软件工程项目表现出高质量；（3）对每个软件项目实施质量控制和质量保证活动；（4）运用度量技术来制定软件过程改进的策略，进而提高最终产品的质量。

人员：软件工程过程中涉及的每个人都要对质量负责。

重要性：要么一次做好，要么重做。如果软件团队在每个软件工程活动中都强调质量，则会减少返工量，进而降低成本，更重要的是可以缩短面市时间。

步骤：在软件质量保证活动启动前，必须

按照多种不同的抽象层次来定义“软件质量”。理解了质量的含义之后，软件团队必须确定一组软件质量保证（SQA）活动来过滤掉工作产品中的错误，以免这些错误再继续传播下去。

工作产品：为了制定软件团队的 SQA 策略，需要建立“软件质量保证计划”。在建模、编码阶段，主要的 SQA 工作产品是技术评审的输出；在测试阶段（第 17～19 章），主要的 SQA 工作产品是制定的测试计划和测试规程。也可能产生其他与过程改进相关的工作产品。

质量保证措施：在错误变成缺陷之前发现它！也就是说，尽量提高缺陷排除效率，进而减少软件团队不得不付出的返工量。

本书所讨论的软件工程方法只有一个目标：在计划时间内生产出高质量的软件。但是很多读者都会遇到这样一个问题：什么是软件质量？

Philip Crosby[Cro79] 在他的有关质量的且颇具影响力的著作中间接地回答了这个问题：

质量管理的问题不在于人们不知道什么是质量，而在于人们自认为知道什么是质量……

每个人都需要质量（当然，是在特定条件下），每个人都觉得自己理解它（尽管人们不愿意解释它），每个人都认为只要顺其自然就可以（毕竟，我们都还做得不错）。当然，大多数人都认为这一领域的问题都是由他人引起的。（只要他们肯花时间就能把事情做好。）

确实，质量是一个具有挑战性的概念，我们已经在第 15 章进行了详细论述。^①

有些软件开发人员仍然相信软件质量是在编码完成之后才应该开始担心的事情。这是完全错误的！软件质量保证（通常称为质量管理）是适用于整个软件过程的一种普适性活动（第 3 章）。

软件质量保证（SQA）包括：（1）SQA 过程；（2）具体的质量保证和质量控制任务（包

① 如果你还没有阅读第 15 章，那么现在应该阅读了。

括技术评审和多层次测试策略); (3) 有效的软件工程实践 (方法和工具); (4) 对所有软件工作产品及其变更的控制 (第 21 章); (5) 保证符合软件开发标准的规程 (在适用的情况下); (6) 测量和报告机制。

本章侧重于管理问题和特定的过程活动, 以使软件组织确保“在恰当的时间以正确的方式做正确的事情”。

16.1 背景问题

对于任何为他人生产产品的企业来说, 质量控制和质量保证都是必不可少的活动。在 20 世纪以前, 质量控制只由生产产品的工匠承担。随着时间推移, 大量生产技术逐渐普及, 质量控制开始由生产者之外的其他人承担。

第一个正式的质量保证和质量控制方案于 1916 年由贝尔实验室提出, 此后迅速风靡整个制造行业。在 20 世纪 40 年代, 出现了更多正式的质量控制方法, 这些方法都将测量和持续的过程改进 [Dem86] 作为质量管理的关键要素。

软件开发质量保证的历史和硬件制造质量的历史同步。在计算机发展的早期 (20 世纪 50 年代和 60 年代), 质量保证只由程序员承担。软件质量保证的标准是 20 世纪 70 年代首先在军方的软件开发合同中出现的, 此后迅速传遍整个商业界的软件开发活动 [IEE93a]。延伸前述的质量定义, 软件质量保证就是为了保证软件高质量而必需的“有计划的、系统化的行动模式” [Sch98c]。质量保证责任的范围最好可以用曾经流行的一个汽车业口号来概括: “质量是头等重要的工作。”对软件来说含义就是, 各个参与者都对软件质量负有责任——包括软件工程师、项目管理者、客户、销售人员和 SQA 小组成员。

SQA 小组充当客户在公司内部的代表。也就是说, SQA 小组成员必须从客户的角度来审查软件。软件是否充分满足第 15 章中提出的各项质量因素? 软件工程实践是否依照预先制定的标准进行? 作为 SQA 活动一部分的技术规范是否恰当地发挥了作用? SQA 小组的工作将回答上述这些问题以及其他问题, 以确保软件质量得到维持。

16.2 软件质量保证的要素

软件质量保证涵盖了广泛的内容和活动, 这些内容和活动侧重于软件质量管理, 可以归纳如下 [Hor03]。

标准。IEEE、ISO 及其他标准化组织制定了一系列广泛的软件工程标准和相关文件。标准可能是软件工程组织自愿采用的, 或者是客户或其他利益相关者责成采用的。软件质量保证的任务是要确保遵循所采用的标准, 并保证所有的工作产品符合标准。

评审和审核。技术评审是由软件工程师执行的质量控制活动, 目的是发现错误。审核是一种由 SQA 人员执行的评审, 意图是确保软件工程工作遵循质量准则。例如, 要对评审过程进行审核, 确保以最有可能发现错误的方式进行评审。

测试。软件测试 (第 17 ~ 19 章) 是一种质量控制功能, 它有一个基本目标——发现错误。SQA 的任务是要确保测试计划适当和实施有效, 以便最有可能实现软件测试的基本

关键概念

软件质量保证的要素
形式化方法
目标
ISO 9001: 2008 标准
质量管理资源
六西格玛
软件可靠性
软件安全
SQA 计划
SQA 任务
统计软件质量保证

引述 你犯了太多错了。

Yogi Berra

网络资源 SQA 的深度探讨 (包含一系列定义) 可在 http://www.swqual.com/images/FoodforThought_Jan2011.pdf 得到。

目标。

错误 / 缺陷的收集和分析。改进的唯一途径是衡量做得如何。软件质量保证人员收集并分析错误和缺陷数据,以便更好地了解错误是如何引入的,以及什么样的软件工程活动最适合消除它们。

变更管理。变更是对所有软件项目最具破坏性的一个方面。如果没有适当的管理,变更可能会导致混乱,而混乱几乎总是导致低质量。软件质量保证将确保进行足够的变更管理实践(第 21 章)。

教育。每个软件组织都想改善其软件工程实践。改善的关键因素是对软件工程师、项目经理和其他利益相关者的教育。软件质量保证组织牵头软件过程改进,并是教育计划的关键支持者和发起者。

供应商管理。可以从外部软件供应商获得三种类型的软件:(1)简易包装软件包(shrink-wrapped package,例如微软 Office);(2)定制外壳(tailored shell)[Hor03]——提供可以根据购买者需要进行定制的基本框架结构;(3)合同软件(contractured software)——按客户公司提供的规格说明定制设计和构建。软件质量保证组的任务是,通过建议供应商应遵循的具体的质量做法(在可能的情况下),并将质量要求作为与任何外部供应商签订合同的一部分,确保高质量的软件成果。

安全防卫。随着网络犯罪和新的关于隐私的政府法规的增加,每个软件组织应制定对策,在各个层面上保护数据,建立防火墙保护 WebApp,并确保软件在内部没有被篡改。软件质量保证确保应用适当的过程和技术来实现软件安全(第 20 章)。

安全。因为软件几乎总是人工设计系统(例如,汽车应用系统或飞机应用系统)的关键组成部分,所以潜在缺陷的影响可能是灾难性的。软件质量保证可能负责评估软件失效的影响,并负责启动那些减少风险所必需的步骤。

风险管理:尽管分析和减轻风险(第 26 章)是软件工程师考虑的事情,但是软件质量保证组应确保风险管理活动适当进行,且已经建立风险相关的应急计划。

除了以上这些问题和活动,软件质量保证还确保将质量作为主要关注对象的软件支持活动(如维护、求助热线、文件和手册)可以高质量地进行和开展。

引述 卓越在于提高所供产品质量的能力是无限的。

Rick Petin

信息栏 质量管理资源

网上有很多质量管理资源,包括专业团体、标准组织和一般的信息资源。从以下网址开始查询是不错的选择:

- 美国质量协会(ASQ)软件分会: www.asq.org/software。
- 美国计算机协会(ACM): www.acm.org。
- 网络安全和信息系统信息分析中心(CSI-AC): <https://sw.thecsiac.com/>。
- 国际标准化组织(ISO): www.iso.ch。
- ISO SPICE: <http://www.spiceusergroup.org/>。

org/。

- 美国波多里奇国家质量奖(Malcolm Baldrige National Quality Award): <http://www.nist.gov/baldrige/>。
- 软件工程研究所: www.sei.cmu.edu/。
- 软件测试和质量工程: www.stickyminds.com。
- 六西格玛资源 www.isixsigma.com/; www.asq.org/sixsigma/。
- TickIT 国际(质量认证主题): www.tic-

kit.org/international.htm。

- 全面质量管理 (TQM): <http://www.isix-sigma.com/methodology/total-quality->

[management-tqm/](http://www.isix-sigma.com/methodology/total-quality-management-tqm/); <http://asq.org/learn-about-quality/total-quality-management/overview/overview.html>。

16.3 软件质量保证的过程和产品特性

当我们开始讨论软件质量保证时,值得注意的是,在一个软件环境中运行良好的 SQA 步骤和方案可能在另一个软件环境中出现问题。即使在采用一致软件工程方法^①的同一个公司,不同的软件产品也可能表现出不同水平的质量 [Par11]。

解决这一困境的方法是在理解软件产品具体的质量需求之后,选取可以用来达到那些需求的过程和具体的 SQA 活动和任务。软件工程研究所的 CMMI 和 ISO 9000 标准是最常用的软件过程框架,两者各提出一套“语法和语义” [Par11],以引导软件工程实践,从而提高产品质量。通过选择框架要素以及使这些要素与某特定产品的质量要求相匹配,软件组织“协调”使用两个模型,而不是实例化一个框架的全部。

16.4 软件质量保证的任务、目标和度量

软件质量保证是由多种任务组成的,这些任务是两种不同的人群相联系的——这两种人群分别是做技术工作的软件工程师和负有质量计划、监督、记录、分析和报告责任的软件质量保证组。

软件工程师通过采用可靠的技术方法和措施,进行技术评审,并进行计划周密的软件测试来获得质量(和执行质量控制活动)。

16.4.1 软件质量保证的任务

软件质量保证组的行动纲领是协助软件团队实现高品质的最终产品。软件工程研究所推荐一套质量保证活动,即质量保证计划、监督、记录、分析和报告。这些活动由独立的软件质量保证组执行(和完成)。

编制项目质量保证计划。该计划作为项目计划的一部分,并经所有利益相关者评审。软件工程组和软件质量保证组进行的质量保证活动都受该计划支配。该计划确定要进行的评估、要进行的审核和评审、适用于项目的标准、错误报告和跟踪的规程、软件质量保证组产出的工作产品以及将提供给软件团队的反馈意见。

参与编写项目的软件过程描述。软件团队选择完成工作的过程。软件质量保证组审查该过程描述是否符合组织方针、内部软件标准、外部要求的标准(例如 ISO-9001),以及是否与软件项目计划的其他部分一致。

评审软件工程活动,以验证其是否符合规定的软件过程。软件质量保证组识别、记录和跟踪偏离过程的活动,并验证是否已做出更正。

审核指定的软件工作产品以验证是否遵守作为软件过程一部分的那些规定。质量保证工作小组审查选定的产品,识别、记录并跟踪偏差,验证

提问 SQA 小组的作用是什么?

引述 质量从来没有意外,它始终是崇高的意图、真诚的努力、聪明的管理和熟练的执行的結果,它表示从多个选项中所做的明智选择。

William A.
Foster

① 例如, CMMI 定义的过程和实践。

已经做出的更正，并定期向项目经理报告其工作成果。

确保根据文档化的规程记录和处理软件工作和工作产品中的偏差。在项目计划、过程描述、适用的标准或软件工程工作产品中可能会遇到偏差。

记录各种不符合项并报告给高层管理人员。跟踪不符合项，直到解决。

除了这些活动，软件质量保证组还协调变更的控制和管理（第 21 章），并帮助收集和分析软件度量。

SafeHome 软件质量保证

[场景] Doug Miller 的办公室，SafeHome 软件项目开始。

[人物] Doug Miller（SafeHome 软件工程项目团队经理）和产品软件工程团队的其他成员。

[对话]

Doug: 非正式评审进行得怎么样了？

Jamie: 我们正在进行项目关键部分的非正式评审，这一部分在测试之前采用结对编程方法，进展比预想的要快。

Doug: 好的，可是我想让 Bridget Thorton 的 SQA 组来审查我们的工作产品，以保证项目是在遵循我们的软件过程，且是符合我们的质量目标的。

Vinod: 他们不是在做大部分的测试吗？

Doug: 是的，但 QA 要比测试做更多的事。我们需要保证文档和代码是一致的，并且保证在集成新构件时不引入错误。

Jamie: 我真的不想由于他们发现的问题而被评头论足。

Doug: 别担心，审核的重点是要检查我们的工作产品是否符合需求，以及活动的过程。我们只会使用审核结果努力改善过程和软件产品。

Vinod: 我相信这将花费更多的时间。

Doug: 从长远来看，当我们尽早发现缺陷时，这将会节省时间，如果早期发现缺陷，修复缺陷也能花费更低的成本。

Jamie: 那么，这听起来是好事。

Doug: 同样重要的是确定哪些活动引入了缺陷，将来增加评审任务来捕获这些缺陷。

Vinod: 这将帮助我们确定我们是否足够仔细地评审活动进行了抽样。

Doug: 我认为长远来看 SQA 活动将使我们成为更好的团队。

16.4.2 目标、属性和度量

执行上节所述的软件质量保证活动，是要实现一套务实的目标。

需求质量。需求模型的正确性、完整性和一致性将对所有后续工作产品的质量有很大的影响。软件质量保证必须确保软件团队严格评审需求模型，以达到高水平的质量。

设计质量。软件团队应该评估设计模型的每个元素，以确保设计模型显示出高质量，并且设计本身符合需求。SQA 寻找能反映设计质量的属性。

代码质量。源代码和相关的工作产品（例如其他说明资料）必须符合本地的编码标准，并具有易于维护的特点。SQA 应该找出那些能合理分析代码质量的属性。

质量控制有效性。软件团队应使用有限的资源，在某种程度上最有可能得到高品质的结果。SQA 分析用于评审和测试上的资源分配，评估其分配方式是否最为有效。

对于所讨论的每个目标，图 16-1[Hya96] 标出了现有的质量属性。可以使用度量数据来

标明所示属性的相对强度。

目标	属性	度量
需求质量	歧义	含糊修饰词的数量 (例如: 许多、大量、与人友好)
	完备性	TBA 和 TBD 的数量
	可理解性	节 / 小节的数量
	易变性	每项需求变更的数量
	可追溯性	变更所需要的时间 (通过活动)
	模型清晰性	不能追溯到设计 / 代码的需求数 UML 模型数 每个模型中描述文字的页数 UML 错误数
设计质量	体系结构完整性	是否存在现成的体系结构模型
	构件完备性	追溯到结构模型的构件数
	接口复杂性	过程设计的复杂性
	模式	挑选一个典型功能或内容的平均数 布局合理性 使用的模式数量
代码质量	复杂性	环路复杂性
	可维护性	设计要素 (第 23 章)
	可理解性	内部注释的百分比
	可重用性	变量命名约定
	文档	可重用构件的百分比 可读性指数
质量控制效率	资源分配	每个活动花费的人员时间百分比
	完成率	实际完成时间与预算完成时间之比
	评审效率	参见评审度量
	测试效率	发现的错误及关键性问题数 改正一个错误所需的工作量 错误的根源

图 16-1 软件质量的目标、属性和度量 [Hya96]

16.5 软件质量保证的形式化方法

前面几节讨论了软件质量是每个人的工作，可以通过出色的软件工程实践以及通过应用技术评审、多层次的测试策略、更好地控制软件工作产品和所做的变更、应用可接受的软件工程标准和过程框架来实现。此外，质量可定义成一组质量属性，并使用各种指标和度量进行（间接）测量。

在过去的 30 年中，软件界有一群人——虽然不多但是很坚决——提出软件质量保证应该采用一种更为形式化的方法。一段计算机程序相当于一个数学对象，每一种程序设计语言都有一套定义严格的语法和语义，而且软件需求规格说明也有严格的方法。如果需求模型（规格说明）和程序设计语言都以严格的方式表示，就可以采用程序正确性证明来说明程序是否严格符合其规格说明。

程序正确性证明并不是什么新的思路。Dijkstra [Dij76a] 和 Linger、Mills、Witt [Lin79]

以及其他很多人都倡导程序正确性证明,并将它与结构化程序设计概念的使用联系在一起(第13章)。

16.6 统计软件质量保证

统计质量保证反映了一种在产业界不断增长的趋势:质量的量化。对于软件而言,统计质量保证包含以下步骤:

1. 收集软件的错误和缺陷信息,并进行分类。
2. 追溯每个错误和缺陷形成的根本原因(例如,不符合规格说明、设计错误、违背标准、缺乏与客户的交流)。
3. 使用 Pareto 原则(80%的缺陷可以追溯到所有可能原因中的20%),将这20%(重要的少数)原因分离出来。
4. 一旦找出这些重要的少数原因,就可以开始纠正引起错误和缺陷的问题。

统计质量保证这个比较简单的概念代表的是创建自适应软件过程的一个重要步骤,在这个过程中要进行修改,以改进那些引入错误的过程元素。

16.6.1 一个普通的例子

举一个例子来说明统计方法在软件工程项目中的应用。假定软件开发组织收集了为期一年的错误和缺陷信息,其中有些错误是在软件开发过程中发现的,另外一些缺陷则是在软件交付给最终用户之后发现的。尽管发现了数以百计的不同问题,但所有问题都可以追溯到下述原因中的一个(或几个):

- 不完整或错误的规格说明(IES)。
- 与客户交流中所产生的误解(MCC)。
- 故意违背规格说明(IDS)。
- 违反程序设计标准(VPS)。
- 数据表示有错(EDR)。
- 构件接口不一致(ICI)。
- 设计逻辑的错误(EDL)。
- 不完整或错误的测试(IET)。
- 不准确或不完整的文档(IID)。
- 将设计转换为程序设计语言实现时的错误(PLT)。
- 不清晰或不一致的人机界面(HCI)。
- 其他(MIS)。

为了使用统计质量保证方法,需要建一张如图16-2所示的表格。表中显示 IES、MCC 和 EDR 即是“重要的少数”,它们导致的错误占错误总数的53%。但是需要注意,在只考虑严重错误时,应该将 IES、EDR、PLT 和 EDL 作为“重要的少数”。一旦确定了这些重要的少数原因,软件开发组织就可以开始采取改正行动了。例如,为了改正 MCC 错误,软件开发者可能要采用需求收集技术(第7章),以提高与客户交流和规格说明的质量。为了改正 EDR 错误,开发者可能要使用工具进行数据建模,并进行更为严格的数据设计评审。

引述 20%的代码含有80%的错误,找到错误,修正错误!
Lowell Arthur

提问 进行统计 SQA 需要哪些步骤?

引述 适当进行的统计分析是对不确定事物的微妙解剖,是对推测的调查。
M. J. Moroney

总计			严重		中等		微小	
错误	数量	百分比	数量	百分比	数量	百分比	数量	百分比
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
总计	942	100%	128	100%	379	100%	435	100%

图 16-2 统计 SQA 数据收集

值得注意的是，改正行动主要是针对“重要的少数”。随着这些“重要的少数”被不断改正，新的“重要的少数”将提到改正日程上来。

已经证明统计软件质量保证技术确实使质量得到了提高（例如 [Rya11]、[Art97]）。在某些情况下，应用这些技术后，软件组织已经取得每年减少 50% 缺陷的好成绩。

统计 SQA 及 Pareto 原则的应用可以用一句话概括：将时间用于真正重要的地方，但是首先你必须知道什么是真正重要的！

16.6.2 软件工程中的六西格玛

六西格玛是目前产业界应用最广泛的基于统计的质量保证策略。20 世纪 80 年代在摩托罗拉公司最先普及，六西格玛策略“是严格且规范的方法学，它运用数据和统计分析，通过识别和消除制造以及服务相关过程中的‘缺陷’来测量和改进企业的运转状况”[ISI08]。“六西格玛”一词来源于六个标准偏差——每百万个操作发生 3.4 个偏差（缺陷），它意味着非常高的质量标准。六西格玛方法学有三个主要的核心步骤：

- 定义：通过与客户交流的方法来定义客户需求、可交付的产品及项目目标。
- 测量：测量现有的过程及其产品，以确定当前的质量状况（收集缺陷度量信息）。
- 分析：分析缺陷度量信息，并挑选出重要的少数原因。

提问 六西格玛方法学的核心步骤是什么？

如果某个现有软件过程是适当的，只是需要改进，则六西格玛还需要另外两个核心步骤：

- 改进：通过消除缺陷根本原因的方式来改进过程。
- 控制：控制过程以保证以后的工作不会再引入缺陷原因。

以上三个核心步骤和另外两个附加步骤有时叫作 DMAIC（定义、测量、分析、改进和控制）方法。

如果某组织正在建立一个软件过程（而不是改进现有的过程），则需要增加下面两个核心步骤：

- 设计：设计过程，以达到：（1）避免缺陷产生的根本原因；（2）满足客户需求。
- 验证：验证过程模型是否确实能够避免缺陷，并且满足客户需求。

上述步骤有时称为 DMADV（定义、测量、分析、设计和验证）方法。

六西格玛的全面讨论不是本书重点，有兴趣的读者可参考 [ISI08]、[Pyz03] 和 [Sne03]。

16.7 软件可靠性

毫无疑问，计算机程序的可靠性是其整个质量的重要组成部分。如果某个程序经常性且反复性地不能执行，那么其他软件质量因素是不是可接受就无所谓了。

与其他质量因素不同，软件可靠性可以通过历史数据和开发数据直接测量和估算出来。按统计术语所定义的软件可靠性是：“在特定环境和特定时间内，计算机程序正常运行的概率” [Mus87]。举个例子来说，如果程序 X 在 8 小时处理时间内的可靠性估计为 0.999，也就意味着，如果程序 X 执行 1000 次，每次运行 8 小时处理时间（执行时间），则 1000 次中正确运行（无失效）的次数可能是 999 次。

无论何时谈到软件可靠性，都会涉及一个关键问题：术语失效（failure）一词是什么含义？在有关软件质量和软件可靠性的任何讨论中，失效都意味着与软件需求的不符。但是这一定义是有等级之分的。失效可能仅仅是令人厌烦的，也可能是灾难性的。有的失效可以在几秒钟之内得到纠正，有的则需要几个星期甚至几个月的时间才能纠正。让问题更加复杂的是，纠正一个失效事实上可能会引入其他的错误，而这些错误最终又会导致其他的失效。

引述 可靠性不得以的代价是简明性。

C. A. R. Hoare

16.7.1 可靠性和可用性的测量

早期的软件可靠性测量工作试图将硬件可靠性理论中的数学公式外推来进行软件可靠性的预测。大多数与硬件相关的可靠性模型依据的是由于“磨损”而导致的失效，而不是由于设计缺陷而导致的失效。在硬件中，由于物理磨损（如温度、腐蚀、振动的影响）导致的失效远比与设计缺陷有关的失效多。不幸的是，软件恰好相反。实际上，所有软件失效都可以追溯到设计或实现问题上，磨损（第 2 章）在这里根本没有影响。

关键点 软件可靠性问题总能追溯到设计或实现中的缺陷。

硬件可靠性理论中的核心概念以及这些核心概念是否适用于软件，对这个问题的争论仍然存在。尽管在两种系统之间尚未建立不可辩驳的联系，但是考虑少数几个同时适用于这两种系统的简单概念却很有必要。

当我们考虑基于计算机的系统时，可靠性的简单测量是平均失效间隔时间（Mean-Time-Between-Failure, MTBF）：

$$MTBF = MTTF + MTTR$$

其中，MTTF（Mean-Time-To-Failure）和 MTTR（Mean-Time-To-Repair）分别是平均失效时间和平均维修时间。^①

关键点 请注意，平均失效间隔时间和相关测量是基于 CPU 时间，而不是挂钟时间。

① 尽管失效可能需要进行调试（和相关的改正），但在大多数情况下，软件不做任何修改，经过重新启动就可以正常工作。

许多研究人员认为 MTBF 是一个远比其他与质量有关的软件度量更有用的测量指标。简而言之,最终用户关心的是失效,而不是总缺陷数。由于一个程序中包含的每个缺陷所具有的失效率不同,因此总缺陷数难以表示系统的可靠性。例如,考虑一个程序,已运行 3000 处理器小时没有发生故障。该程序中的许多缺陷在被发现之前可能有数万小时都不会被发现。隐藏如此深的错误的 MTBF 可能是 30000 甚至 60000 处理器小时。其他尚未发现的缺陷,可能有 4000 或 5000 小时的失效率。即使第一类错误(那些具有长时间的 MTBF)都去除了,对软件可靠性的影响也是微不足道的。

然而,MTBF 可能会产生问题的原因有两个:(1)它突出了失效之间的时间跨度,但不会为我们提供一个凸显的失效率;(2)MTBF 可能被误解为平均寿命,即使这不是它的含义。

可靠性的另一个可选衡量指标是失效率(Failures-In-Time, FIT)——一个部件每 10 亿机时发生多少次失效的统计测量。因此,1FIT 相当于每 10 亿机时发生一次失效。

除可靠性测量之外,还应该进行可用性测量。软件可用性是指在某个给定时间点上程序能够按照需求执行的概率。其定义为:

$$\text{可用性} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})} \times 100\%$$

MTBF 可靠性测量对 MTTF 和 MTTR 同样敏感。而可用性测量在某种程度上对 MTTR 较为敏感,MTTR 是对软件可维护性的间接测量。有关软件可靠性测量的更广泛讨论可参看 [Laz11]。

建议 可用性的某些方面(这里不讨论)与失效没有关系。例如,计划停机(用于技术支持功能)也使软件不可用。

16.7.2 软件安全

软件安全是一种软件质量保证活动,它主要用来识别和评估可能对软件产生负面影响并促使整个系统失效的潜在灾难。如果能够在软件过程的早期阶段识别出这些灾难,就可以指定软件设计特性来消除或控制这些潜在的灾难。

引述 民众的安全应是最崇高的法律。

Cicero

建模和分析过程可以视为软件安全的一部分。开始时,根据危险程度和风险高低对灾难进行识别和分类。例如,与汽车上的计算机巡航控制系统相关的灾难可能有:(1)产生失去控制的加速,不能停止;(2)踩下刹车踏板后没有反应(关闭);(3)开关打开后不能启动;(4)减速或加速缓慢。一旦识别出这些系统级的灾难,就可以运用分析技术来确定这些灾难发生的严重性和概率^①。为了达到高效,应该将软件置于整个系统中进行分析。例如,一个微小的用户输入错误(人也是系统的组成部分)有可能会被软件错误放大,产生将机械设备置于不正确位置的控制数据,此时当且仅当外部环境条件满足时,机械设备的不正确位置将引发灾难性的失效。失效树分析、实时逻辑及 Petri 网模型等分析技术 [Eri05] 可以用于预测可能引起灾难的事件链,以及事件链中的各个事件出现的概率。

引述 我无法想象是什么情况可能使该船沉没。现代造船术已经不存在这种可能了。

E. I. Smith

(泰坦尼克船长)

一旦完成了灾难识别和分析,就可以进行软件中与安全相关的需求规格说明了。在规格说明中包括一张不希望发生的事件清单,以及针对这些事件所希望产生的系统响应。这样就指明了软件在管理不希望发生的事件方面应起的作用。

① 这个方法与第 26 章介绍的风险分析方法类似,主要区别是它注重技术问题,而不是项目相关的问题。

尽管软件可靠性和软件安全彼此紧密相关,但是弄清它们之间的微妙差异非常重要。软件可靠性使用统计分析的方法来确定软件失效发生的可能性,而失效的发生未必导致灾难或灾祸。软件安全则考察失效导致灾难发生的条件。也就是说,不能在真空中考虑失效,而是要在整个计算机系统及其所处环境的范围内加以考虑。

软件安全的全面讨论超出了本书的范围,对软件安全和相关系统问题有兴趣的读者可参考 [Fir12]、[Har12]、[Smi05] 和 [Lev95]。

网络资源 关于软件安全的有价值的论文集可以在 www.safeware-eng.com 找到。

16.8 ISO 9000 质量标准^①

质量保证体系可以定义为:用于实现质量管理的组织结构、责任、规程、过程和资源 [ANS87]。创建质量保证体系的目的是帮助组织以符合规格说明的方式,保证组织的产品和服务满足客户的期望。这些体系覆盖了产品整个生命周期的多种活动,包括计划、控制、测量、测试和报告,并在产品的开发和制造过程中改进质量等级。ISO 9000 标准以通用的术语描述了质量保证体系要素,能够适用于任何行业——不论提供的是何种产品或服务。

某个公司要注册成为 ISO 9000 质量保证体系中的一种模式,该公司的质量体系和实施情况应该由第三方的审核人员仔细检查,查看其是否符合标准以及实施是否有效。成功注册之后,这个公司将获得由审核人员所代表的注册登记实体颁发的证书。此后每半年进行一次监督审核,以此保证该公司的质量体系持续符合标准。

ISO 9001:2008 中描述的质量要求涉及管理者责任、质量体系、合同评审、设计控制、文件和资料控制、产品标识与可追溯性、过程控制、检验和试验、纠正及预防措施、质量记录的控制、内部质量审核、培训、服务以及统计技术等主题。软件组织要登记为 ISO 9001:2008 认证,就必须针对上述每个方面的质量要求(以及其他方面的质量要求)制定相关的政策和规程,并且有能力证明组织活动的确是按照这些政策和规程实施的。希望进一步了解有关 ISO 9001:2008 信息的读者可参考 [Coc11]、[Hoy09] 或 [Cia09]。

网络资源 大量有关 ISO 9000/9001 资源的链接可在 www.tantara.ab.ca/info.htm 找到。

信息栏 | ISO 9001:2008 标准

下面的大纲定义了 ISO 9001:2008 标准的基本要素。与标准有关的详细信息可从国际标准化组织 (www.iso.ch) 及其他网络信息源(如 www.praxiom.com) 找到。

- 建立质量管理体系的要素。
 - 建立、实施和改进质量体系。
 - 制定质量方针,强调质量体系的重要性。
- 编制质量体系文件。
 - 描述过程。
 - 编制操作手册。
 - 制定控制(更新)文件的方法。
 - 制定记录保存的方法。
- 支持质量控制和质量保证。
 - 提高所有利益相关者对质量重要性的认识。
 - 注重客户满意度。
 - 制定质量计划来描述目的、职责和权力。

① 本节由 Michael Stovsky 编写,根据“Fundamentals of ISO 9000”改编,这是由 R.S.Pressman & Associates, Inc. 为音像教程“Essential Software Engineering”开发的工作手册。使用已得到授权。

- 制定所有利益相关者间的交流机制。
- 为质量管理体系建立评审机制。
 - 确定评审方法和反馈机制。
 - 制定跟踪程序。
- 确定质量资源（包括人员、培训、基础设施要素）。
- 建立控制机制。
 - 针对计划。
- 针对客户需求。
- 针对技术活动（如分析、设计、试验）。
- 针对项目监测和管理。
- 制定补救措施。
 - 评估质量数据和度量。
 - 为持续的过程和质量改进制定措施。

16.9 软件质量保证计划

SQA 计划为软件质量保证提供了一张路线图。该计划由 SQA 小组（如果没有 SQA 小组，则由软件团队）制定，作为各个软件项目中 SQA 活动的模板。

IEEE 已经公布了 SQA 计划的标准 [IEE93]。该标准建议 SQA 计划应包括：（1）计划的目的和范围；（2）SQA 覆盖的所有软件工程工作产品的描述（例如，模型、文档、源代码）；（3）应用于软件过程中的所有适用的标准和习惯做法；（4）SQA 活动和任务（包括评审和审核）以及它们在整个软件过程中的位置；（5）支持 SQA 活动和任务的工具和方法；（6）软件配置管理（第 29 章）的规程；（7）收集、保护和维护所有 SQA 相关记录的方法；（8）与产品质量相关的组织角色和责任。

软件工具 软件质量管理

[目标] 使用 SQA 工具的目的是辅助项目团队评估和改进软件工作产品的质量。

[机制] 工具的机制各异。一般情况下，其目的是评估特定工作产品的质量。注意，SQA 工具类通常包含很多软件测试工具（第 17～19 章）。

[代表性工具]^①

- QA Complete。由 SmartBear (<http://smartbear.com/products/qa-tools/test-management>) 开发，QA 管理确保在软件开发过程的每一阶段做到完整的测试覆盖。
- QPR Suite。由 QPR Software (<http://www.qpr.com>) 开发，提供对六西格玛及其他质量管理方法的支持。
- Quality Tools and Templates。由 iSixSigma (<http://www.isixsigma.com/tools-templates/>) 开发，描述了大量有用的质量管理工具和方法。
- NASA Quality Resources。由 Goddard Space Flight Center (<http://www.hq.nasa.gov/office/codeq/software/ComplexElectronics/checklists.htm>) 开发，提供了有用的 SQA 表格、模板、检查单和工具。

16.10 产品度量框架

测量将数字或符号赋给现实世界中的实体属性。为达到这个目标，需要一个包含统一规

① 这里记录的工具并不代表本书支持这些工具，而只是这类工具的例子。在大多数情况下，工具的名字由各自的开发者注册为商标。

则的测量模型。尽管测量理论（例如，[Kyb84]）及其在计算机软件中的应用（例如，[Zus97]）等主题超出了本书的讨论范围，但是，为测量软件的产品度量建立一个基本框架和一组基本原则是值得的。

引述 一门科学与其测量工具一样成熟。

Louis Pasteur

16.10.1 测度、度量和指标

尽管测量（measurement）、测度（measure）和度量（metric）这三个术语常常可以互换使用，但注意到它们之间的细微差别是很重要的。由于“测度”一词可用作名词也可用作动词，因此，这个术语的定义是令人费解的。在软件工程中，“测度”为产品或过程的某些属性的范围、数量、维度、容量或大小提供量化的指标。而“测量”是确定测度的动作。度量在《软件工程术语的 IEEE 标准词汇表》[IEE93b] 中的定义为：度量是一个系统、构件或过程具有给定属性的量化测量程度。

提问 测度和测量有什么不同？

当收集了一个数据点时（例如，在一个软件构件中发现的错误数），就已建立了一个测度。收集一个或多个数据点（例如，考察一些构件评审和单元测试，以收集每个单元测试错误数的测度）就产生了测量。软件度量以某种方式（例如，每次评审发现错误的平均数，或每个单元测试所发现错误的平均数）与单个测度相关。

关键点 一个指标就是提供了对过程、产品或项目深入理解的一个或一些度量。

软件工程师收集测度并开发度量以便获得指标。一个指标是一个度量或多个度量的组合，它提供了对软件过程、软件项目或产品本身的深入理解。指标提供的理解使项目经理或软件工程师能够调整过程、项目或产品以使其变得更好。

16.10.2 产品度量的挑战

在过去 40 年中，许多研究人员试图开发出单一的度量值，以为软件复杂性提供全面的测量。Fenton[Fen94] 将这种研究描绘为“寻找不可能的圣杯”。尽管已提出了许多复杂性测量 [Zus90]，但每种方法都对什么是复杂性以及哪些系统属性导致复杂性持有不同的看法。作为类比，我们考虑用来评价汽车吸引力的度量，一些观察者可能强调车身设计，另一些会强调机械特性，还有一些鼓吹价格、性能、燃料经济性或汽车报废时的回收能力。由于这些特性中的任意一个都有可能和其他特性不一致，因此，很难为“吸引力”制定一个单一值。对计算机软件而言，会出现同样的问题。

网络资源 美国国土安全部汇编了关于产品度量的大量信息，网址是 <http://buildsecuri-tyin.us-cert.gov/bsi/articles/best-practices/measurement.html>。

然而，仍有必要去测量和控制软件的复杂度。如果一个度量的单一值难以获取，那么可能应该开发针对不同内部程序属性（例如，有效模块化、功能独立性和第 11 章论及其他属性）的测度。这些测量和由此产生的度量可用作需求模型和设计模型的独立指标。但是新问题再次出现，Fenton[Fen94] 说道：“尝试去寻找刻画许多不同属性的测度是危险的，其危险性在于，测度不可避免地必须满足有冲突的目标。这与测量的代表性理论是相反的。”尽管 Fenton 所说的是正确的，但许多人提出，在软件过程的早期阶段执行的产品测量为软件工程师评估质量^①提供了一致和客观的机制。

① 虽然文献中对具体度量的评论是常见的，但许多评论关注深奥的问题而忽略了现实世界中度量的主要目标：帮助软件工程师建立一种系统、客观的方法，来获得对其工作的深入理解，并最终提高产品的质量。

16.10.3 测量原则

在介绍一系列的产品度量之前,重要的是理解基本的测量原则。产品度量的作用主要包括:(1)辅助分析模型和设计模型的评估;(2)提供过程设计和源代码复杂性的指示;(3)方便更有效测试的设计。Roche[Roc94]提出了能用以下5个活动描述的测量过程:公式化、收集、分析、解释、反馈。软件度量仅当被有效地特征化并经过确认以证明它们的价值时才是有用的。人们已经提出了许多度量特征化和确认原则,其中一些有代表性的原则如下[Let03b]:

- 度量应该具有符合要求的数学特性。也就是说,度量的值应该在有意义的范围之内(例如0~1,其中0意味着不存在,1表示最大值,0.5表示中间点)。同时,所谓在合理范围内的度量不应该仅由顺序测量的成分组成。
- 如果度量代表一个软件特征,当正向品质出现时特征值提高,当不理想品质出现时特征值下降,那么度量值提高或降低的方式应当是一致的。
- 每种度量在发布或用于做决策之前,应该在广泛的环境中根据经验加以确认。度量应该测量重要的、与其他因素无关的因素。它应该扩展到大系统中,并能在许多程序设计语言和系统领域中行得通。

尽管公式化、特征化和确认很关键,但收集和分析才是驱动测量过程的活动。Roche[Roc94]为这些活动提供了以下指导原则:(1)只要有可能,数据的收集与分析应能自动化地进行;(2)应该使用有效的统计技术以建立内部产品属性与外部质量特性之间的关系(例如,体系结构的复杂程度是否与产品使用报告中的缺陷数相关);(3)应该为每个度量建立解释性指导原则和建议。

16.10.4 面向目标的软件测量

目标/问题/度量(Goal/Question/Metric, GQM)范型是由Basili和Weiss[Bas84]开发的,这种技术用于为软件过程的任何部分识别出有意义的度量。GQM强调了以下要求:(1)确定特定过程活动的明确测量目标或将要评估的产品特性;(2)定义一组必须回答的问题以达到目标;(3)确定一些良好定义的度量以帮助我们回答这些问题。

目标定义模板[Bas94]可用于定义每个测量目标。模板采取以下形式:

在…{进行测量的环境}…环境中,从…{对测量感兴趣的人}…的角度,关于…{所考虑的活动或属性}…方面,为…{分析的总体目标^①}…目的,分析…{将要测量的属性和活动名}…。

作为一个例子,考虑SafeHome的目标定义模板:

在未来三年要对产品进行改进的环境下,从软件工程师完成工作的角度,关于SafeHome具有较强可扩展能力方面,为了评估体系结构构件,分析SafeHome软件体系结构。

明确定义了测量目标之后,形成一组问题。回答这些问题有助于软件团队(或其他利益相关者)确定是否已达到测量目标。可能会问到的问题如下:

引述 就像温度测量开始于一根食指……后来逐渐增长到复杂的级别、工具和技术。软件测量的成熟也是如此。

Shari Pfleeger

网络资源 GQM的有益讨论可在www.thecsiac.com/resources/ref_documents/software-acquisition-goal-practice-goal-question-metric-gqm-approach找到。

^① van Solingen 和 Berghout [Sol99] 建议:目标几乎总是“理解、控制或改善”过程活动或产品属性。

问题 1: 体系结构构件是否将以功能与相关数据分开的方式描述?

问题 2: 每个构件的复杂性是限定在一定的范围内以便于修改与扩展吗?

每个问题都应该利用一个或多个测度和度量以量化的方式回答。例如, 度量若给出了体系结构构件内聚性(第 11 章)指标, 则可能对回答问题 1 有用。本章后面讨论的度量有助于理解问题 2。在每种情况下, 所选择(或派生)的度量应该与 16.10.3 节所讨论的测量原则和 16.10.5 节所讨论的测量属性相符。

16.10.5 有效软件度量的属性

尽管已提出了数百种计算机软件度量, 但不是所有的度量都为软件工程师提供了实用的支持。一些度量所要求的测量太复杂, 一些度量太深奥以致现实世界很少有专业人员能够理解, 另一些度量则违背了高质量软件的直观概念。

Ejiogu[Eji91] 定义了一组有效软件度量所应具有的属性。导出的度量及导出度量的测度应该是容易学习的, 且其计算不应该需要过多的工作量或时间。度量应该满足工程师对所考虑的产品属性的直观看法(例如, 测度模型内聚的度量在数值上应随内聚等级的增长而增长)。度量产生的结果应该总是无歧义的。度量的数学计算应该使用不会导致奇异单位组合的测度。例如, 项目组成员使用多种编程语言就会导致可疑的单位组合, 这样就没有直观的说服力。度量应该基于需求模型、设计模型或程序结构本身, 而不应该依赖于变化多样的编程语言的语法或语义。度量应该提供信息以产生高质量的最终产品。

尽管大多数软件度量满足这些属性, 但一些常用的度量可能不满足其中某种属性。例如功能点方法——一种软件交付功能的测量。有人提出独立的第三方与其同事就算用相同的软件信息也不可能导出同样的功能点值[⊖]。难道我们应该由此拒绝使用功能点度量吗? 当然不能! 功能点提供了有用的观点且由此提供了不同的数值, 尽管它不能很好地满足某个属性。

建议 经验表明, 只有产品度量是直观的且易于计算时, 才会得到使用。若需要大量的“计数”, 且需要复杂的计算, 则该度量不可能得到广泛采纳。

SafeHome 关于产品度量的辩论

[场景] Vinod 的工作间。

[人物] Vinod、Jamie、Ed, SafeHome 软件工程团队的成员, 他们正在进行构件级设计和测试用例设计。

[对话]

Vinod: Doug(Doug Miller, 软件工程经理)告诉我, 我们都应该使用产品度量, 但他有点含糊, 又说他不强迫这件事情, 是否采用由我们自己决定。

Jamie: 那好。我们没有时间做测量工作, 我们都在为维持进度而奋战。

Ed: 我同意 Jamie 的观点。我们的确面临这个问题, 我们没时间。

Vinod: 是的, 我知道, 但是使用产品度量可能会有一些好处。

Jamie: 我并不怀疑这个问题, Vinod, 这是时间问题, 我没有任何剩余时间来做产品度量。

Vinod: 但是, 如果测量能节省你的时间, 那又怎么样呢?

Ed: 你错了, 就像 Jamie 所说的那样, 需要时间……

⊖ 可以提出类似的有力的辩论。这是软件度量的本质。

Vinod: 不, 等等, 如果它能节省我们的时间, 那又怎么样呢?

Jamie: 你说呢?

Vinod: 返工……正是这样。如果我们使用的一个度量有助于我们避免一个主要的或一个中等的问题, 那它就节省了返工一部分系统所需要的时间, 我们节省了时间。不是吗?

Ed: 我想这有可能, 但你能保证某个产品度量能帮助我们找到问题吗?

Vinod: 你能保证它不能帮助我们找到问题吗?

Jamie: 那你计划怎么办?

Vinod: 我认为我们应该选择一些设计度量, 可能是面向类的, 将它们作为我们所开发的每个评审过程的一部分。

Ed: 我确实不太熟悉面向类的度量。

Vinod: 我花一些时间查查, 然后给一些建议。怎么样, 你们这些家伙?

(Ed 和 Jamie 淡漠地点点头)

习题与思考题

- 16.1 有人说“差异控制是质量控制的核心”, 既然每个程序都与其他程序互不相同, 我们应该寻找什么样的差异? 应该如何控制这些差异?
- 16.2 如果客户不断改变他想做的事情, 是否还有可能评估软件质量?
- 16.3 质量和可靠性是两个相关的概念, 但在许多方面却有根本的不同, 请就此进行讨论。
- 16.4 一个正确的程序还能不可靠吗? 请加以解释。
- 16.5 一个正确的程序可能表现不出高质量吗? 请加以解释。
- 16.6 为什么软件工程小组与独立的软件质量保证小组之间的关系经常是紧张的? 这种紧张关系是否是正常的?
- 16.7 假设赋予你改进组织中软件质量的职责, 你要做的第一件事是什么? 然后呢?
- 16.8 除了可以统计错误和缺陷之外, 还有哪些可以统计的软件特性是具有质量意义的? 它们是什么? 能否直接测量?
- 16.9 软件中的 MTBF 概念仍然不断受到批评, 请解释为什么。
- 16.10 给出两个安全性至关重要的计算机控制系统。并为每个系统列出至少三项与软件失效直接相关的灾难。
- 16.11 获取一份 ISO 9001:2000 和 ISO 9000-3^① 的副本, 准备一次讨论作业, 讨论三个方面的 ISO 9001 质量要求及其在软件行业中是如何应用的。

扩展阅读与信息资源

关于规范的计算机软件质量保证规程, Chemuturi(《Mastering Software Quality Assurance》, J.Ross Publishing, 2010)、Hoyle(《Quality Management Essentials》, Butterworth-Heinemann, 2007)、Tian(《Software Quality Engineering》, Wiley-IEEE Computer Society Press, 2005)、El Emam(《The ROI from Software Quality》, Auerbach, 2005)、Horch(《Practical Guide to Software Quality Management》, Artech House, 2003)以及Nance和Arthur(《Managing Software Quality》, Springer, 2002)的书为我们展现了什么是优秀的管理水平。Deming[Dem86]、Defoe和Juran(《Juran's Quality Handbook》, 6th ed., McGraw-Hill, 2010)、Juran(《Juran on Quality by Design》, Free Press, 1992)以及Crosby[Cro79]和《Quality Is Still Free》, McGraw-Hill, 1995)的书虽然不是软件方面的书, 但是对于负责软件开发

① 标准 ISO 9000-3 为 ISO 9001:2000 在计算机软件的使用指南, 现已改版为 ISO 90003。——译者注

的高级经理来说是必读的。Gluckman 和 Roome (《Everyday Heroes of the Quality Movement》, Dorset House, 1993) 通过讲述质量过程中参与者的一个故事, 把质量问题人性化了。Kan (《Metrics and Models in Software Quality Engineering》, Addison-Wesley, 1995) 提出了软件质量的量化观点。

Evans (《Quality & Performance Excellence》, South-Western College Publishing, 2007 和《Total Quality: Management, Organization and strategy》, 4th ed., South-Western College Publishing, 2004)、Bru (《Six Sigma for Managers》, McGraw-Hill, 2005) 和 Dobb (《ISO9001:2000 Quality Registration Step-by-Step》, 3rd ed., Butterworth-Heinemann, 2004) 的书分别是许多关于 TQM、六西格码和 ISO 9001:2000 的书籍中的代表。

O'Connor 和 Kleyner (《Practical Reliability Engineering》, Wiley, 2012)、Naik 和 Tripathy (《Software Testing and Quality Assurance: Theory and Practice》, Wiley-Spektrum, 2008)、Pham (《System Software Reliability》, Springer, 2006)、Musa (《Software Reliability Engineering: More Reliable Software, Faster Development and Testing》, 2nd ed., McGraw-Hill, 2004) 以及 Peled (《Software Reliability Methods》, Springer, 2001) 撰写了介绍测量和分析软件可靠性方法的实用指南。

Vincoli (《Basic Guide to System Safety》, Wiley, 2006)、Dhillon (《Computer System Reliability, Safety and Usability》, CRC Press, 2013 和《Engineering Safety》, World Scientific Publishing Co., 2003) 以及 Hermann (《Software Safety and Reliability》, Wiley-IEEE Computer Society Press, 2010), 还有 Verma、Ajit 和 Karanki (《Reliability and Safety Engineering》, Springer, 2010)、Storey (《Safety-Critical Computer Systems》, Addison-Wesley, 1996) 以及 Leveson [Lev95] 的书是目前出版的最全的讨论软件安全和系统安全的书籍。此外, van der Meulen (《Definitions for Hardware and Software Safety Engineers》, Springer-Verlag, 2000) 提供了一份完整的可靠性和安全性方面重要概念和术语的汇编, Gardiner (《Testing Safety-Related Software》, Springer-Verlag, 1999) 提供了测试安全关键系统的专业指导。Friedman 和 Voas (《Software Assessment: Reliability Safety and Testability》, Wiley, 1995) 提供了评估可靠性和安全性的有用模型。Ericson (《Hazard Analysis Primer》, CreateSpace Independent Publishing Platform, 2012 和《Hazard Analysis Techniques for System Safety》, Wiley, 2005) 讲述了日益重要领域的公害分析。

网上有软件质量保证和相关主题的大量信息资源。SQA 相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 的“software engineering resources”下找到。

软件测试策略

要点浏览

概念：软件测试的目的是为了发现软件设计和实现过程中因疏忽所造成的错误。但是，如何进行测试？是否应该制定正式的测试计划？应该将整个程序作为一个整体来测试，还是应该只测试其中的一小部分？当向一个大型系统加入新的构件时，对于已经做过的测试，是否还要重新测试？什么时候需要客户参与测试工作？在制定测试策略时，就需要回答上述问题以及一些其他问题。

人员：软件测试策略由项目经理、软件工程师及测试专家来制定。

重要性：测试所花费的工作量经常比其他任何软件工程活动都多。若测试是无计划进行的，则既浪费时间，又浪费不必要的劳动。甚至更糟的是，错误未被测出，因而隐蔽下来。因此，为测试软件建立系统化的测试策略是合情合理的。

步骤：测试从“小范围”开始，并逐步过

渡到“软件整体”。这意味着，早期的测试关注单个构件或相关的一小组构件，利用测试发现封装在构件中的数据错误和处理逻辑错误。当完成单个构件的测试后，需要将构件集成起来，直到建成整个系统。这时，执行一系列的高阶测试以发现不满足客户需求的错误。随着错误的发现，必须利用调试过程对错误进行诊断和纠正。

工作产品：测试规格说明是将软件测试团队的具体测试方法文档化。这主要包括制定描述整体策略的计划，定义特定测试步骤的规程以及将要进行测试的类型。

质量保证措施：在测试进行之前通过对测试规格说明进行评审，可以评估测试用例及测试任务的完整性。有效的测试计划和规程可以引导团队有秩序地构建软件，并且在构建过程中能够发现各个阶段引入的错误。

软件测试策略提供了一张路线图：描述将要进行的测试步骤，包括这些步骤的计划和执行时机，以及需要的工作量、时间和资源。因此，任何测试策略都必须包含测试计划、测试用例设计、测试执行以及测试结果数据的收集与评估。

软件测试策略应该具有足够的灵活性，以促进测试方法的定制；同时又必须足够严格，以支持在项目进行过程中对项目进行合理策划和追踪管理。Shooman[SHO83]对这些问题进行了讨论：

从许多方面来看，测试都是一个独立的过程，并且同软件开发方法一样，测试类型也有很多种。多年以来，对付程序出错的唯一武器就是谨慎的设计和程序员的个人智慧。目前，有很多现代设计技术（和正式技术评

关键概念

- α 测试
- β 测试
- 自底向上集成
- 类测试
- 簇测试
- 完成
- 配置评审
- 调试
- 部署测试
- 驱动
- 独立测试组
- 集成测试

审)可以帮助我们减少代码中内在的初始错误。类似地,不同的测试方法正在逐渐聚合成几种不同的途径和思想。

这些途径和思想就是我们所谓的策略。本章讨论软件测试策略,本书的第 18、19 章介绍实施测试策略的测试方法和测试技术。

17.1 软件测试的策略性方法

测试是可以事先计划并可以系统地进行的一系列活动。因此,应该为软件过程定义软件测试模板,即将特定的测试用例设计技术和测试方法放到一系列的测试步骤中去。

文献中已经提出了许多软件测试策略。这些策略为软件开发人员提供了测试模板,且具备下述一般特征:

- 为完成有效的测试,应该进行有效的、正式的技术评审(第 20 章)。通过评审,许多错误可以在测试开始之前排除。
- 测试开始于构件层,然后向外“延伸”到整个基于计算机系统的集成中。
- 不同的测试技术适用于不同的软件工程方法和不同的时间点。
- 测试由软件开发人员和(对大型项目而言)独立的测试组执行。
- 测试和调试是不同的活动,但任何测试策略都必须包括调试。

软件测试策略必须提供必要的低级测试,可以验证小段源代码是否正确实现,也要提供高级测试,用来确认系统的主要功能是否满足用户需求。软件测试策略必须为专业人员提供工作指南,同时,为管理者提供一系列的里程碑。由于测试策略的步骤往往是在软件完成的最后期限的压力开始呈现时才刚刚进行的,因此,测试的进度必须是可测量的,并且应该让问题尽可能早地暴露。

17.1.1 验证与确认

软件测试是通常所讲的更为广泛的主题——验证与确认(Verification and Validation, V&V)的一部分。验证是指确保软件正确地实现某一特定功能的一系列活动,而确认指的是确保开发的软件可追溯到客户需求的另外一系列活动。Boehm[BOE81]用另一种方式说明了这两者的区别:

验证:“我们在正确地构建产品吗?”

确认:“我们在构建正确的产品吗?”

验证与确认的定义包含很多软件质量保证活动(第 16 章)^①。

验证与确认包含广泛的 SQA 活动:正式技术评审、质量和配置审核、性能监控、仿真、可行性研究、文档评审、数据库评审、算法分析、开发测试、易用性测试、合格性测试、验收测试和安装测试。虽然测试在验证与确认中起到了非常重要的作用,但很多其他活动也是必不可少的。

关键概念

面向对象软件
性能测试
恢复测试
回归测试
安全测试
冒烟测试
压力测试
桩
系统测试
基于线程的测试
自顶向下集成
单元测试确认
确认测试
验证

网络资源

有关软件测试的有用资料可在 www.mtsu.edu/~storm/ 找到。

引述

测试是开发软件系统过程中每项可靠的工作都不可避免的部分。

William Howden

^① 应该注意到,哪些类型的测试构成“确认”,然而人们对此观点存在极大的分歧。一些人认为所有的测试都是验证,而确认是在对需求进行评审和认可时进行的,也许更晚一些——是在系统投入运行时由用户进行的。另外一些人将单元测试和集成测试(17.3.1 节和 17.3.2 节)看成验证,而将高阶测试(17.5 节)看成确认。

测试确实为软件质量的评估（更实际地说是错误的发现）提供了最后的堡垒。但是，测试不应当被看作安全网。正如人们所说的那样：“你不能测试质量。如果开始测试之前质量不佳，那么当你完成测试时质量仍然不佳。”在软件工程的整个过程中，质量已经被包含在软件之中了。方法和工具的正确运用、有效的正式技术评审、坚持不懈的管理与测量，这些都形成了在测试过程中所确认的质量。

Miller[MIL77]将软件测试和质量保证联系在一起，他认为：“无论是大规模系统还是小规模系统，程序测试的根本动机都是使用经济且有效的方法来确认软件质量。”

17.1.2 软件测试组织

对每个软件项目而言，在测试开始时就会存在固有的利害关系冲突。要求开发软件的人员对该软件进行测试，这本身似乎是没有恶意的！毕竟，谁能比开发者本人更了解程序呢？遗憾的是，这些开发人员感兴趣的是急于显示他们所开发的程序是无错误的，是按照客户的需求开发的，而且能按照预定的进度和预算完成。这些利害关系会影响软件的充分测试。

从心理学的观点来看，软件分析和设计（连同编码）是建设性的任务。软件工程师分析、建模，然后编写计算机程序及其文档。与其他任何建设者一样，软件工程师也为自己的“大厦”感到骄傲，而蔑视企图拆掉大厦的任何人。当测试开始时，有一种微妙的但确实存在的企图，即试图摧毁软件工程师所建造的大厦。以开发者的观点来看，可以认为（心理学上）测试是破坏性的。因此，开发者精心地设计和执行测试，试图证明其程序的正确性，而不是注意发现错误。遗憾的是，错误仍然是存在的，而且，即使软件工程师没有找到错误，客户也会发现它们。

上述讨论通常会使人产生下面的误解：（1）软件开发人员根本不应该做测试；（2）应当让那些无情地爱挑毛病的陌生人做软件测试；（3）测试人员仅在测试步骤即将开始时参与项目。这些想法都是不正确的。

软件开发人员总是要负责程序各个单元（构件）的测试，确保每个单元完成其功能或展示所设计的行为。在多数情况下，开发者也进行集成测试。集成测试是一个测试步骤，它将给出整个软件体系结构的构建（和测试）。只有在软件体系结构完成后，独立测试组才开始介入。

独立测试组（Independent Test Group, ITG）的作用是为了避免开发人员进行测试所引发的固有问题。独立测试可以消除利益冲突。独立测试组的成员毕竟是依靠找错误来获得报酬的。

然而，软件开发人员并不是将程序交给独立测试组就可以一走了之。在整个软件项目中，开发人员和测试组要密切配合，以确保进行充分的测试。在测试进行的过程中，必须随时可以找到开发人员，以便及时修改发现的错误。

从分析与设计到计划和制定测试规程，ITG参与整个项目过程。从这种意义上讲，ITG是软件开发项目团队的一部分。然而，在很多情况下，

建议 不要轻易地将测试看成是一个安全网，认为它能捕捉由不良的软件工程实践引发的所有错误。应该在整个软件过程中注重质量和错误检测。

引述 乐观主义是编程的职业障碍；测试是治疗良方。

Kent Beck

关键点 独立测试组没有软件开发者可能经历的“利益冲突”。

引述 人们犯的第一个错误是认为测试团队负责保证质量。

Brian Marick

ITG 直接向软件质量保证组织报告，由此获得一定程度的独立性。如果 ITG 是软件工程团队的一部分，那么这种独立性将是不可能获得的。

17.1.3 软件测试策略——宏观

可以将软件过程看作图 17-1 所示的螺旋。开始时系统工程定义软件的角色，从而引出软件需求分析，在需求分析中建立了软件的信息域、功能、行为、性能、约束和确认标准。沿着螺旋向内，经过设计阶段，最后到达编码阶段。为开发计算机软件，沿着流线螺旋前进，每走一圈都会降低软件的抽象层次。

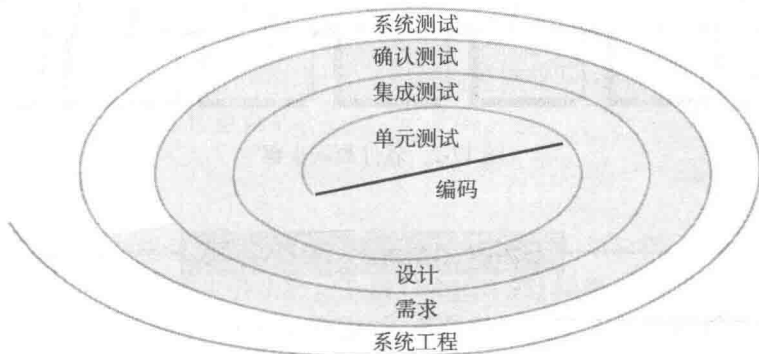


图 17-1 测试策略

软件测试策略也可以放在螺旋模型中来考虑（图 17-1）。单元测试始于螺旋的旋涡中心，侧重于以源代码形式实现的每个单元（例如，构件、类或 WebApp 内容对象）。沿着螺旋向外就是集成测试，这时的测试重点在于软件体系结构的设计和构建。沿着螺旋向外再走一圈就是确认测试，在这个阶段，依据已经建立的软件，对需求（作为软件需求建模的一部分而建立）进行确认。最后到达系统测试阶段，将软件与系统的其他成分作为一个整体来测试。为了测试计算机软件，沿着流线向外螺旋前进，每转一圈都拓宽了测试范围。

提问 什么是软件测试的总体策略？

以过程的观点考虑整个测试过程，软件工程环境中的测试实际上就是按顺序实现四个步骤，如图 17-2 所示。最初，测试侧重于单个构件，确保它起到了单元的作用，因此称之为单元测试。单元测试充分利用测试技术，运行构件中每个控制结构的特定路径，以确保路径的完全覆盖，并最大限度地发现错误。接下来，组装或集成各个构件以形成完整的软件包。

网络资源 有关软件测试人员的有用资源可在站点 www.SQAtester.com 中找到。

集成测试处理并验证与程序构建相关的问题。在集成过程中，普遍使用关注输入和输出的测试用例设计技术（尽管也使用检验特定程序路径的测试用例设计技术来保证主要控制路径的覆盖）。在软件集成（构建）完成之后，要执行一系列的高阶测试。必须评估确认准则（需求分析阶段建立的）。确认测试为软件满足所有的功能、行为和性能需求提供最终保证。

最后的高阶测试步骤已经超出软件工程的边界，属于更为广泛的计算机系统工程范围。软件一旦确认，就必须与其他系统成分（如硬件、人、数据库）结合在一起。系统测试验证所有成分都能很好地结合在一起，且能满足整个系统的功能或性能需求。

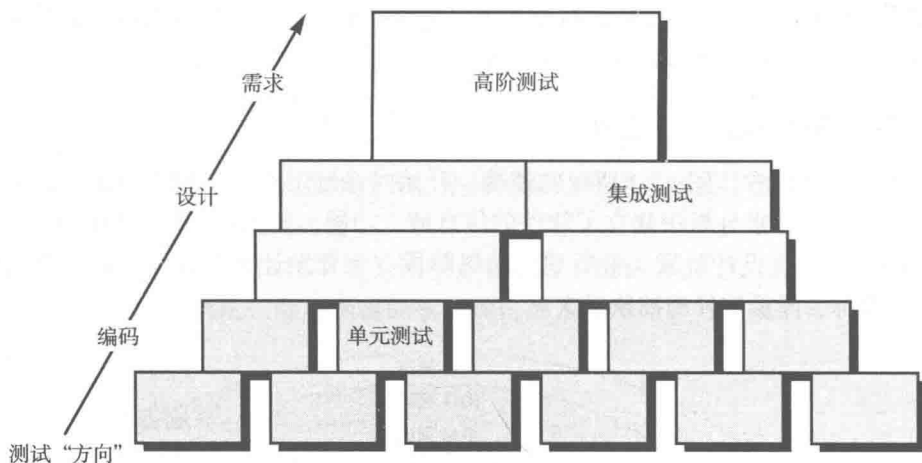


图 17-2 软件测试步骤

SafeHome 准备测试

[场景] Doug Miller 的办公室，继续构件级设计，并开始特定构件的构建。

[人物] Doug Miller，软件工程师；Vinod、Jamie、Ed 和 Shakira，SafeHome 软件工程团队成员。

[对话]

Doug: 在我看来，我们似乎是没有花费太多的时间讨论测试。

Vinod: 对，但我们都有点忙。另外，我们一直在考虑这个问题……实际上，远不止考虑。

Doug (微笑): 我知道，大家都在超负荷地工作，不过我们还得全面考虑。

Shakira: 在对构件开始编码之前，我喜欢设计单元测试，因此，那是我一直以来尽力去做的。我有一个相当大的测试文件，一旦完成了构件编码工作，就运行这个测试文件。

Doug: 那是极限编程（敏捷软件开发过程，

见第 5 章）概念，不是吗？

Ed: 是的。尽管我没有亲自使用极限编程，但可以肯定，在建立构件之前设计单元测试是个好主意，这种单元测试的设计会给我们提供所需要的所有信息。

Jamie: 我一直在做这件事情。

Vinod: 我负责集成，因此，每当别人将构件传给我，我就将其集成到部分已集成的程序中，并运行一系列的集成测试。我一直忙于为系统中的每个功能设计适当的测试集。

Doug (对 Vinod): 你多长时间运行一次测试？

Vinod: 每天……直到系统被集成……嗯，直到我们计划交付的软件增量被集成。

Doug: 你们已经走在我前面了。

Vinod (大笑): 在软件业务中，抢先就是一切，老板。

17.1.4 测试完成的标准

每当讨论软件测试时，就会引出一个典型的问题：“测试什么时候才算做完？怎么知道我们已做了足够的测试？”非常遗憾的是，这个问题没

提问 我们什么时候完成测试？

有确定的答案，只是有一些实用的答复和早期的尝试可作为经验指导。

对上述问题的一个答复是：你永远也不能完成测试，这个担子只会从你（软件工程师）身上转移到最终用户身上。用户每次运行计算机程序时，程序就在经受测试。这个严酷的事实突出了其他软件质量保证活动的重要性。另一个答复（有点讽刺意味，但无疑是准确的）是：当你的时间或资金耗尽时，测试就完成了。

尽管很少有专业人员对上面的答复有异议，但软件工程师还是需要更严格的标准，以确定充分的测试何时能做完。净室软件工程方法提出了统计使用技术 [kel100]：运行从统计样本中导出的一系列测试，统计样本来自目标群的所有用户对程序的所有可能执行。通过在软件测试过程中收集度量数据并利用现有的统计模型，对于回答“测试何时做完”这种问题，还是有可能提出有意义的指导性原则的。

引述 仅仅针对最终用户需求进行测试，就如同在牺牲了地基、大梁及管道工程的情况下，只根据内部设计师已经完成的工作对建筑进行检查一样。

Boris Beizer

17.2 策略问题

本章的后面几节介绍系统化的软件测试策略。然而，如果忽视了一些重要问题，即使最好的策略也会失败。Tom Gilb[GIL95] 提出，只有软件测试人员解决了下述问题，软件测试策略才会获得成功：（1）早在开始测试之前，就要以量化的方式规定产品需求；（2）明确地陈述测试目标；（3）了解软件的用户并为每类用户建立用户描述；（4）制定强调“快速周期测试”的测试计划；^①（5）建立能够测试自身的“健壮”软件（防错技术在 17.3.1 节讨论）；（6）测试之前，利用有效的正式技术评审作为过滤器；（7）实施正式技术评审以评估测试策略和测试用例本身；（8）为测试过程建立一种持续的改进方法。

提问 什么样的指导原则使软件测试策略获得成功？

网络资源 相当好的测试资源列表可在 www.SQAtester.com 找到。

17.3 传统软件的测试策略^②

许多策略可用于测试软件。其中的一个极端是，软件团队等到系统完全建造后再对整个系统执行测试，以期望发现错误。虽然这种方法很有吸引力，但效果不好，可能得到的是有许多缺陷的软件，致使所有的利益相关者感到失望。另一个极端是，无论系统的任何一部分在何时建成，软件工程师每天都在进行测试。

多数软件团队选择介于这两者之间的测试策略。这种策略以渐进的观点对待测试，以个别程序单元的测试为起点，逐步转移到便于单元集成的测试（有的时候每天都进行测试），最后以实施整个系统的测试而告终。下面几节将对这几种不同的测试进行描述。

建议 在为构件开发代码之前就设计单元测试用例是个不错的想法，有助于确保开发的代码能够通过测试。

17.3.1 单元测试

单元测试侧重于软件设计的最小单元（软件构件或模块）的验证工作。利用构件级设计

① Gilb[Gil95] 建议软件团队学习对客户可用性进行快速周期测试（项目工作的 2%），至少在“可试验性”方面增强功能性或提高质量。从这些快速周期测试得到的反馈可以用于控制质量级别及相应的测试策略。

② 本书使用术语常规软件和传统软件来表示在多种应用领域中经常碰到的普通分层软件体系结构或调用-返回软件体系结构。传统的软件体系结构不是面向对象的，也不包括 WebApp 或移动 App。

描述作为指南,测试重要的控制路径以发现模块内的错误。测试的相对复杂度和这类测试发现的错误受到单元测试约束范围的限制。单元测试侧重于构件的内部处理逻辑和数据结构。这种类型的测试可以对多个构件并行执行。

单元测试问题。图 17-3 对单元测试进行了概要描述。测试模块的接口是为了保证被测程序单元的信息能够正常地流入和流出;检查局部数据结构以确保临时存储的数据在算法的整个执行过程中能维持其完整性;执行控制结构中的所有独立路径(基本路径)以确保模块中的所有语句至少执行一次;测试边界条件确保模块在到达边界值的极限或受限处理的情形下仍能正确执行。最后,要对所有的错误处理路径进行测试。

对穿越模块接口的数据流的测试要在任何其他测试开始之前进行。若数据不能正确地输入/输出,则其他测试都是没有意义的。另外,应当测试局部数据结构,可能的话,在单元测试期间确定对全局数据的局部影响。

在单元测试期间,选择测试的执行路径是最基本的任务。设计测试用例是为了发现因错误计算、不正确的比较或不适当的控制流而引起的错误。

边界测试是最重要的单元测试任务之一。软件通常在边界处出错,也就是说,错误行为往往出现在处理 n 维数组的第 n 个元素,或者 i 次循环的第 i 次调用,或者遇到允许出现的最大、最小数值时。使用刚好小于、等于或大于最大值和最小值的数据结构、控制流和数值作为测试用例就很有可能发现错误。

好的设计要求能够预置出错条件并设置异常处理路径,以便当错误确实出现时重新确定路径或彻底中断处理。Yourdon[YOU75]称这种方法为防错法(antibugging)。遗憾的是,存在的一种趋势是在软件中引入异常处理,然而却从未对其进行测试。如果已经实现了错误处理路径,就一定要对其进行测试。

在评估异常处理时,应能测试下述的潜在错误:(1)错误描述难以理解;(2)记录的错误与真正遇到的错误不一致;(3)在异常处理之前,错误条件就引起了操作系统的干预;(4)异常条件处理不正确;(5)错误描述没有提供足够的信息,对确定错误产生原因没有帮助。

单元测试过程。单元测试通常被认为是编码阶段的附属工作。可以在编码开始之前或源代码生成之后进行单元测试的设计。设计信息的评审可以指导建立测试用例,发现前面所讨论的各类错误,每个测试用例都应与一组预期结果联系在一起。

由于构件并不是独立的程序,因此,必须为每个测试单元开发驱动程序和桩程序。单元测试环境如图 17-4 所示。在大多数应用中,驱动程序只是一个“主程序”,它接收测试用例数据,将这些数据传递给(将要测试的)构件,并打印相关结果。桩程序的作用是替换那些从属于被测构件(或被其调用)的模块。桩程序或“伪程序”使用从属模块的接口,可能做少量



图 17-3 单元测试

提问 单元测试期间常发现的错误是什么?

建议 确信已经设计了执行每个异常处理路径的测试。若没有,当执行这样的路径时就可能失败,从而加重了危险的形势。

建议 在没有资源做全面测试的情况下,只选择关键模块和高复杂性模块做单元测试。

的数据操作，提供入口的验证，并将控制返回到被测模块。

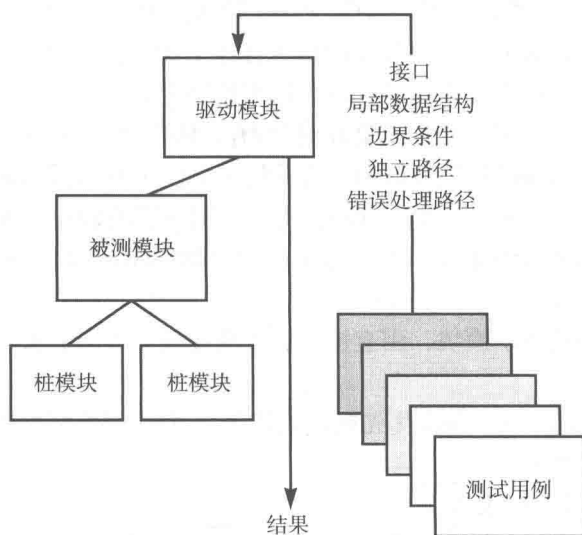


图 17-4 单元测试环境

驱动程序和桩程序都意味着测试开销。也就是说，两者都必须编写代码（通常并没有使用正式的设计），但并不与最终的软件产品一起交付。若驱动程序和桩程序保持简单，实际开销就会比较低。遗憾的是，在只使用“简单”的驱动程序和桩程序的情况下，许多构件是不能完成充分的单元测试的，因此，完整的测试可以延迟到集成测试这一步（这里也要使用驱动程序和桩程序）。

17.3.2 集成测试

软件界的初学者一旦完成所有模块的单元测试之后，可能会问一个似乎很合理的问题：如果每个模块都能单独工作得很好，那么为什么要怀疑将它们放在一起时的工作情况呢？当然，这个问题涉及“将它们放在一起”的接口连接。数据可能在穿过接口时丢失；一个模块可能对另一个模块产生负面影响；子功能联合在一起并不能达到预期的功能；单个模块中可以接受的不精确性在连接起来之后可能会扩大到无法接受的程度；全局数据结构可能产生问题。遗憾的是，问题还远不止这些。

建议 采取一步到位的集成方法是一种懒惰的策略，注定会失败。在进行测试时，应该采用增量集成。

集成测试是构建软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试。其目标是利用已通过单元测试的构件建立设计中描述的程序结构。

常常存在一种非增量集成的倾向，即利用“一步到位”的方式来构造程序。所有的构件都事先连接在一起，全部程序作为一个整体进行测试。结果往往是一片混乱！会出现一大堆错误。由于在整个程序的广阔区域中分离出错的原因是非常复杂的，因此改正错误也会比较困难。

增量集成与“一步到位”的集成方法相反。程序以小增量的方式逐步进行构建和测试，这样错误易于分离和纠正，更易于对接口进行彻底测试，而且可以运用系统化的测试方法。下面将讨论一些不同的增量集成策略。

自顶向下集成。自顶向下集成测试是一种构建软件体系结构的增量方法。模块的集成顺

序为从主控模块（主程序）开始，沿着控制层次逐步向下，以深度优先或广度优先的方式将从属于（和间接从属于）主控模块的模块集成到结构中去。

参见图 17-5，深度优先集成是首先集成位于程序结构中主控路径上的所有构件。主控路径的选择有一点武断，也可以根据特定应用的特征进行选择。例如，选择最左边的路径，首先集成构件 M_1 、 M_2 和 M_5 。其次，集成 M_8 或 M_6 （若 M_2 的正常运行是必需的），然后集成中间和右边控制路径上的构件。广度优先集成首先沿着水平方向，将属于同一层的构件集成起来。如图 17-5 中，首先将构件 M_2 、 M_3 和 M_4 集成起来，其次是下一个控制层 M_5 、 M_6 ，依此类推。集成过程可以通过下列 5 个步骤完成：

- 1. 主控模块用作测试驱动模块，用直接从属于主控模块的所有模块代替桩模块。
 - 2. 依靠所选择的集成方法（深度优先或广度优先），每次用实际模块替换一个从属桩模块。
 - 3. 集成每个模块后都进行测试。
 - 4. 在完成每个测试集之后，用实际模块替换另一个桩模块。
 - 5. 可以执行回归测试（在本节的后面讨论）以确保没有引入新的错误。
- 回到第 2 步继续执行此过程，直到完成了整个程序结构的构建。

建议 制定项目进度时，必须考虑将要采取的集成方式，使得构件在需要时是可用的。

提问 自顶向下集成的步骤是什么？

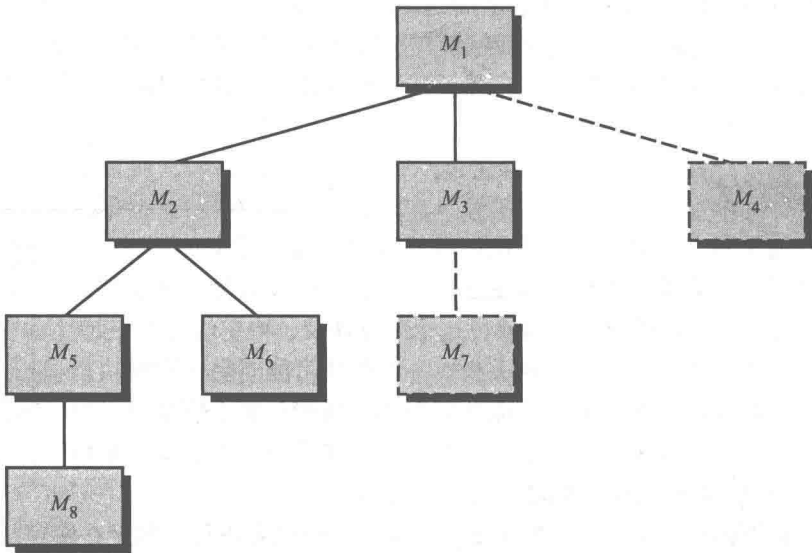


图 17-5 自顶向下集成

自顶向下集成策略是在测试过程的早期验证主要控制点或决策点。在能够很好分解的程序结构中，决策发生在层次结构的较高层，因此会首先遇到。如果主控问题确实存在，尽早地发现是有必要的。若选择了深度优先集成方法，可以实现和展示软件的某个完整功能。较早的功能展示可以增强所有开发者、投资者及用户的信心。

自底向上集成测试。自底向上集成测试，顾名思义，就是从原子模块（程序结构的最底层构件）开始进行构建和测试。由于构件是自底向上集成的，在处理时所需要的从属于给定层次的模块总是存在的，因此，没有必

提问 选择自顶向下集成方法时，可能会遇到什么问题？

要使用桩模块。自底向上集成策略可以利用以下步骤来实现：

1. 连接低层构件以构成完成特定子功能的簇（有时称为构造）。
2. 编写驱动模块（测试的控制程序）以协调测试用例的输入和输出。
3. 测试簇。
4. 去掉驱动程序，沿着程序结构向上逐步连接簇。

遵循这种模式的集成如图 17-6 所示。连接相应的构件形成簇 1、簇 2 和簇 3，利用驱动模块（图中的虚线框）对每个簇进行测试。簇 1 和簇 2 中的构件从属于模块 M_a ，去掉驱动模块 D_1 和 D_2 ，将这两个簇直接与 M_a 相连。与之相类似，在簇 3 与 M_b 连接之前去掉驱动模块 D_3 。最后将 M_a 和 M_b 与构件 M_c 连接在一起，依此类推。

提问 自底向上的集成步骤是什么？

关键点 自底向上集成排除了对复杂桩的需要。

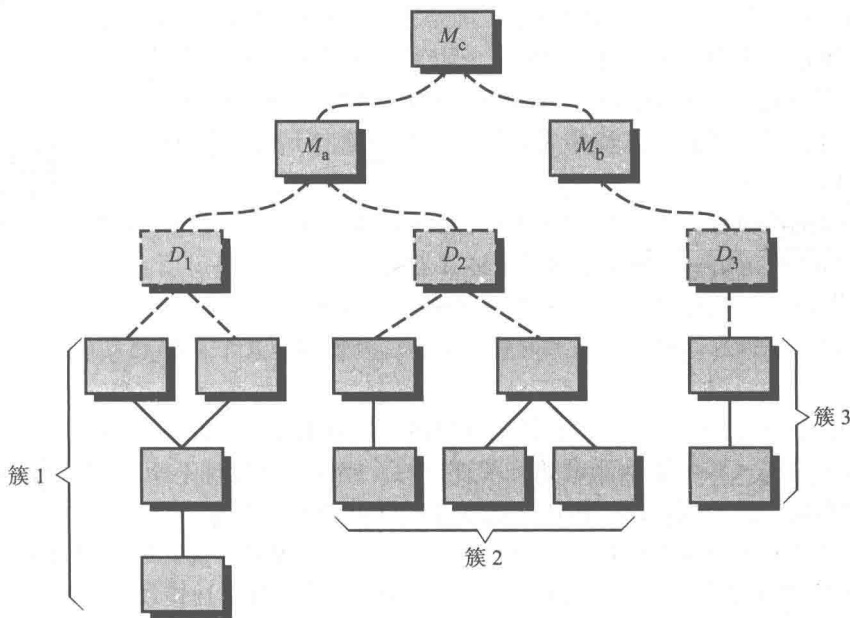


图 17-6 自底向上集成

随着集成的向上进行，对单独的测试驱动模块的需求减少。事实上，若程序结构的最上两层是自顶向下集成的，则驱动模块的数量可以大大减少，而且簇的集成得到了明显简化。

回归测试。每当加入一个新模块作为集成测试的一部分时，软件发生变更，建立了新的数据流路径，可能出现新的 I/O，还可能调用新的控制逻辑。这些变更所带来的副作用可能会使原来可以正常工作的功能产生问题。在集成测试策略的环境下，回归测试是重新执行已测试过的某些子集，以确保变更没有传播不期望的副作用。回归测试有助于保证变更（由于测试或其他原因）不引入无意识行为或额外的错误。

回归测试可以手工进行，方法是重新执行所有测试用例的子集，或者利用捕捉 / 回放工具自动进行。捕捉 / 回放工具使软件工程师能够为后续的回放与比较捕捉测试用例和测试结果。回归测试套件（将要执行的测试子集）包含以下三种测试用例：

- 能够测试软件所有功能的具有代表性的测试样本。
- 额外测试，侧重于可能会受变更影响的软件功能。

建议 回归测试是减少“副效应”的重要方法。每次对软件做重要变更时（包括新构件的集成），都要进行回归测试。

- 侧重于已发生变更的软件构件测试。

随着集成测试的进行,回归测试的数量可能变得相当庞大,因此,应将回归测试套件设计成只包括涉及每个主要程序功能的一个或多个错误类的测试。

冒烟测试。开发软件产品时,冒烟测试是一种常用的集成测试方法,是时间关键项目的决定性机制,允许软件团队频繁地对项目进行评估。大体上,冒烟测试方法包括下列活动。

1. 将已经转换为代码的软件构件集成到构造 (build) 中。一个构造包括所有的数据文件、库、可复用的模块以及实现一个或多个产品功能所需的工程化构件。
2. 设计一系列测试以暴露影响构造正确地其功能的错误。其目的是发现极有可能造成项目延迟的业务阻塞 (show stopper) 错误。
3. 每天将该构造与其他构造及整个软件产品 (以其当前的形式) 集成起来进行冒烟测试。这种集成方法可以是自顶向下的,也可以自底向上的。

每天频繁的测试让管理者和专业人员都能够对集成测试的进展做出实际的评估。McConnell[MCO96] 是这样描述冒烟测试的:

冒烟测试应该对整个系统进行彻底的测试。它不一定是穷举的,但应能暴露主要问题。冒烟测试应该足够彻底,以使得若构造通过测试,则可以假定它足够稳定以致能经受更彻底的测试。

当应用于复杂的、时间关键的软件工程项目时,冒烟测试提供了下列好处:

- 降低了集成风险。冒烟测试是每天进行的,能较早地发现不相容性和业务阻塞错误,从而降低了因发现错误而对项目进度造成严重影响的可能性。
- 提高最终产品的质量。由于这种方法是面向构建 (集成) 的,因此,冒烟方法既有可能发现功能性错误,也有可能发现体系结构和构件级设计错误。若较早地改正了这些错误,产品的质量就会更好。
- 简化错误的诊断和修正。与所有的集成测试方法一样,冒烟测试期间所发现的错误可能与新的软件增量有关,也就是说,新发现的错误可能来自刚加入到构造中的软件。
- 易于评估进展状况。随着时间的推移,更多的软件被集成,也更多地展示出软件的工作状况。这就提高了团队的士气,并使管理者对项目进展有较好的把握。

集成测试工作产品。软件集成的总体计划和特定的测试描述应该在测试规格说明中文档化。这项工作产品包含测试计划和测试规程,并成为软件配置的一部分。测试可以分为若干个阶段和处理软件特定功能及行为特征的若干个构造来实施。例如, SafeHome 安全系统的集成测试可以划分为以下测试阶段:用户交互,传感器处理,通信功能及警报处理。

每个集成测试阶段都刻画了软件内部广泛的功能类别,而且通常与软件体系结构中特定的领域相关,因此,对应于每个阶段建立了相应的程序构造 (模块集)。

集成的进度、附加的开发以及相关问题也在测试计划中讨论。确定每个阶段的开始和结束时间,定义单元测试模块的“可用性窗口”。附加软件 (桩模块及驱动模块) 的简要描述侧重于可能需要特殊工作的特征。最后,描述测试环境和资源。特殊的硬件配置、特殊的仿

关键点 冒烟测试被称为是一种滚动的集成测试方法。每天对软件进行重构 (加入新的构件), 并进行冒烟测试。

引述 将每日构造当成项目的心跳。如果没有了心跳, 项目就死了。

Jim McCarthy

提问 从冒烟测试中可以得到什么好处?

真器和专门的测试工具或技术也是需要讨论的问题。

紧接着需要描述的是实现测试计划所必需的详细测试规程。描述集成的顺序以及每个集成步骤中对应的测试,其中也包括所有的测试用例(带注释以便后续工作参考)和期望的结果列表。

实际测试结果、问题或特例的历史要记录在测试报告中,若需要的话可附在测试规格说明后面。这部分包含的信息在软件维护期间很重要。也要给出适当的参考文献和附录。

17.4 面向对象软件的测试策略^①

简单地说,测试的目标就是在现实的时间范围内利用可控的工作量找到尽可能多的错误。对于面向对象软件,尽管这个基本目标是不变的,但面向对象软件的本质特征改变了测试策略和测试战术(第 19 章)。

17.4.1 面向对象环境中的单元测试

在考虑面向对象的软件时,单元的概念发生了变化。封装导出了类和对象的定义。这意味着每个类和类的实例包装有属性(数据)和处理这些数据的操作。封装的类常是单元测试的重点,然而,类中包含的操作(方法)是最小的可测试单元。由于类中可以包含很多不同的操作,且特殊的操作可以作为不同类的一部分存在,因此,必须改变单元测试的战术。

我们不再孤立地对单个操作进行测试(传统的单元测试观点),而是将其作为类的一部分。为便于说明,考虑一个类层次结构,在此结构内对超类定义某操作 X ,并且一些子类继承了操作 X 。每个子类使用操作 X ,但它应用于为每个子类定义的私有属性和操作的环境内。由于操作 X 应用的环境有细微的差别,因此有必要在每个子类的环境中测试操作 X 。这意味着在面向对象环境中,以独立的方式测试操作 X (传统的单元测试方法)往往是无效的。

面向对象软件的类测试等同于传统软件的单元测试。不同的是传统软件的单元测试侧重于模块的算法细节和穿过模块接口的数据,而面向对象软件的类测试是由封装在该类中的操作和类的状态行为驱动的。

17.4.2 面向对象环境中的集成测试

由于面向对象软件没有明显的层次控制结构,因此,传统的自顶向下和自底向上集成策略(17.3.2 节)已没有太大意义。另外,由于类的成分间直接或间接的相互作用,因此每次将一个操作集成到类中(传统的增量集成方法)往往是不可能的[Ber93]。

面向对象系统的集成测试有两种不同的策略[Bin94b]。一种策略是基于线程的测试(thread-based testing),对响应系统的一个输入或事件所需的一组类进行集成。每个线程单独地集成和测试。应用回归测试以确保没有产生副作用。另一种方法是基于使用的测试(use-based testing),通过测试很少使用服务类(如果有的话)的那些类(称为独立类)开始系统的构

关键点 面向对象的类测试与传统软件的模块测试相似。对操作进行孤立测试是不可取的。

关键点 面向对象软件集成测试的一个重要策略是基于线程的测试。线程是对一个输入或事件做出反应的类集合。基于使用的测试侧重于那些不与其他类进行频繁协作的类。

^① 基本的面向对象概念在附录 2 中介绍。

建。独立类测试完成后，利用独立类测试下一层次的类（称为依赖类）。继续依赖类的测试直到完成整个系统。

在进行面向对象系统的集成测试时，驱动模块和桩模块的使用也发生了变化。驱动模块可用于低层操作的测试和整组类的测试。驱动模块也可用于代替用户界面，以便在界面实现之前就可以进行系统功能的测试。桩模块可用于类间需要协作但其中的一个或多个协作类还未完全实现的情况。

簇测试（cluster testing）是面向对象软件集成测试中的一个步骤。这里，借助试图发现协作错误的测试用例来测试（通过检查 CRC 和对象关系模型所确定的）协作的类簇。

17.5 确认测试

确认测试始于集成测试的结束，那时已测试完单个构件，软件已组装成完整的软件包，且接口错误已被发现和改正。在进行确认测试或系统级测试时，不同类型软件之间的差别已经消失，测试便集中于用户可见的动作和用户可识别的系统输出。

确认可用几种方式进行定义，但是，其中一个简单（尽管粗糙）的定义是当软件可以按照客户合理的预期方式工作时，确认就算成功。在这一点上，喜欢吹毛求疵的软件开发人员可能会提出异议：“谁或者什么是合理预期的裁决者呢？”如果已经开发了软件需求规格说明文档，那么此文档就描述了所有用户可见的软件属性，并包含确认准则部分，确认准则部分就形成了确认测试方法的基础。

关键点 与所有其他测试步骤类似，确认测试尽力发现错误，但是它侧重于需求级的错误，即那些对最终用户而言显而易见的错误。

17.5.1 确认测试准则

软件确认是通过一系列表明软件功能与软件需求相符合的测试而获得的。测试计划列出将要执行的测试类，测试规程定义了特定的测试用例，设计的特定测试用例用于确保软件满足所有的功能需求，具有所有的行为特征，所有内容都准确无误且正确显示，达到所有的性能需求，文档是正确的、可用的，且满足其他需求（如可移植性、兼容性、错误恢复和可维护性）。如果发现了与规格说明的偏差，则要创建缺陷列表。并且必须确定（利益相关者可以接受的）解决缺陷的方法。

17.5.2 配置评审

确认过程的一个重要成分是配置评审。评审的目的是确保所有的软件配置元素已正确开发、编目，且具有改善支持活动的必要细节。有时将配置评审称为审核（audit），这将在第 21 章详细讨论。

17.5.3 α 测试和 β 测试

对软件开发者而言，预见用户如何实际使用程序几乎是不可能的。软件使用指南（使用手册）可能会被错误理解；可能会使用令用户感到奇怪的数据组合；测试者看起来很明显的输出对于工作现场的用户却是难以理解的。

为客户开发定制软件时，执行一系列验收测试能使客户确认所有的需求。验收测试是由最终用户而不是软件工程师进行的，它的范围从非正式的“测试驱动”直到有计划地、系统

地进行一系列测试。实际上,验收测试的执行可能超过几个星期甚至几个月,因此,可以发现长时间以来影响系统的累积错误。

若将软件开发为产品,由多个用户使用,让每个用户都进行正式的验收测试,这当然是不切实际的。多数软件开发者使用称为 α 测试与 β 测试的过程,以期查找到似乎只有最终用户才能发现的错误。

α 测试是由有代表性的最终用户在开发者的场所进行。软件在自然设置下使用,开发者站在用户的后面观看,并记录错误和使用问题。 α 测试在受控的环境下进行。

β 测试在一个或多个最终用户场所进行。与 α 测试不同,开发者通常不在场,因此, β 测试是在不为开发者控制的环境下“现场”应用软件。最终用户记录测试过程中遇见的所有问题(现实存在的或想象的),并定期报告给开发者。接到 β 测试的问题报告之后,开发人员对软件进行修改,然后准备向最终用户发布软件产品。

β 测试的一种变体称为客户验收测试,有时是按照合同交付给客户时进行的。客户执行一系列的特定测试,试图在从开发者那里接收软件之前发现错误。在某些情况下(例如,大公司或政府系统),验收测试可能是非常正式的,会持续很多天甚至好几个星期。

引述 只要给予足够的关注,所有 bug 都是容易找到的(例如:给予足够多的 β 测试人员和相关的开发人员,几乎每个问题都能很快捕获,且容易修改)。

E. Raymond

提问 α 测试和 β 测试之间的区别是什么?

SafeHome 准备确认

[场景] Doug Miller 的办公室,构件级设计及这些构件的构建工作正继续进行。

[人物] Doug Miller, 软件工程师; Vinod、Jamie、Ed 和 Shakira, SafeHome 软件工程团队成员。

[对话]

Doug: 我们将在三个星期内准备好第一个增量的确认,怎么样?

Vinod: 大概可以吧。集成进展得不错。我们每天执行冒烟测试,找到了一些 bug,但还没有我们处理不了的事情。到目前为止,一切都很好。

Doug: 跟我谈谈确认。

Shakira: 可以。我们将使用所有的用例作为测试设计的基础。目前我还没有开始,但我将为我负责的所有用例开发测试。

Ed: 我这里也一样。

Jamie: 我也一样。但是我们已经将确认测试与 α 测试和 β 测试一起考虑了,不是吗?

Doug: 是,事实上我一直考虑请外包商帮我们做确认测试。在预算中我们有这笔钱……它将给我们新的思路。

Vinod: 我认为确认测试已经在我们的控制之中了。

Doug: 我确信是这样,但 ITG (独立测试组)能用另一种眼光来看这个软件。

Jamie: 我们的时间很紧了,Doug,我没有时间培训新人来做这项工作。

Doug: 我知道,我知道。但 ITG 仅根据需求和用例来工作,并不需要太多的培训。

Vinod: 我仍然认为确认测试已经在我们的控制之中了。

Doug: 我知道,Vinod,但在这方面我将强制执行。计划这周的后几天与 ITG 见面。让他们开始工作并看他们有什么意见。

Vinod: 好的,或许这样做可以减轻工作负荷。

17.6 系统测试

在本书的开始,我们就强调过,软件只是基于计算机大系统的一部分。最终,软件要与其他系统成分(如硬件、人和信息)相结合,并执行一系列集成测试和确认测试。这些测试已超出软件过程的范围,而且不仅仅由软件工程师执行。然而,软件设计和测试期间所采取的步骤可以大大提高在大系统中成功地集成软件的可能性。

引述 与死亡和税收一样,测试既是令人不愉快的,也是不可避免的。

Ed Yourdon

一个传统的系统测试问题是“相互指责”。这种情况出现在发现错误时,每个系统成分的开发人员都因为这个问题抱怨别人。其实大家都不应该陷入这种无谓的争论之中,软件工程师应该预见潜在的接口问题,以及:(1)设计出错处理路径,用以测试来自系统其他成分的所有信息;(2)在软件接口处执行一系列模拟不良数据或其他潜在错误的测试;(3)记录测试结果,这些可作为出现“相互指责”时的“证据”;(4)参与系统测试的计划和设计,以保证软件得到充分的测试。

17.6.1 恢复测试

多数基于计算机的系统必须从错误中恢复并在一定的时间内重新运行。在有些情况下,系统必须是容错的,也就是说,处理错误绝不能使整个系统功能都停止。而在有些情况下,系统的错误必须在特定的时间内或严重的经济危害发生之前得到改正。

恢复测试是一种系统测试,通过各种方式强制地让系统发生故障,并验证其能适当恢复。若恢复是自动的(由系统自身完成),则对重新初始化、检查点机制、数据恢复和重新启动都要进行正确性评估。若恢复需要人工干预,则应计算平均恢复时间(Mean-Time-To-Repair, MTTR)以确定其值是否在可接受的范围之内。

17.6.2 安全测试

任何管理敏感信息或能够对个人造成不正当伤害(或带来好处)的计算机系统都是非礼或非法入侵的目标。入侵包括广泛的活动:黑客为了娱乐而试图入侵系统,不满的雇员为了报复而试图破坏系统,不良分子在非法利益驱使下试图入侵系统。

安全测试验证建立在系统内的保护机制是否能够实际保护系统不受非法入侵。引用Beizer[Bei84]的话来说:“系统的安全必须经受住正面的攻击,但是也必须能够经受住侧面和背后的攻击。”

只要有足够的时间和资源,好的安全测试最终将能够入侵系统。系统设计人员的作用是使攻破系统所付出的代价大于攻破系统之后获取信息的价值。安全测试和安全工程在第20章详细讨论。

17.6.3 压力测试

本章前面所讨论的软件测试步骤能够对正常的程序功能和性能进行彻底的评估。压力测试的目的是使软件面对非正常的情形。本质上,进行压力测试的测试人员会问:“在系统失效之前,能将系统的运行能力提高到什么程度?”

压力测试要求以一种非正常的数量、频率或容量的方式执行系统。例如:(1)在平均每秒出现1~2次中断的情形下,可以设计每秒产生10次中断的测试用例;(2)将输入数据

的量提高一个数量级以确定输入功能将如何反应；(3) 执行需要最大内存或其他资源的测试用例；(4) 设计可能在实际的运行系统中产生惨败的测试用例；(5) 创建可能会过多查找磁盘驻留数据的测试用例。从本质上来说，压力测试者将试图破坏程序。

压力测试的一个变体称为敏感性测试。在一些情况下（最常见的是在数学算法中），包含在有效数据界限之内的一小部分数据可能会引起极端处理情况，甚至是错误处理或性能的急剧下降。敏感性测试试图在有效输入类中发现可能会引发系统不稳定或者错误处理的数据组合。

引述 若你正在尽力查找实际系统的 bug，且没有为你的软件提供实际的压力测试，那么现在应该不是你立即开始的时候了。

Boris Beizer

17.6.4 性能测试

对于实时和嵌入式系统，提供所需功能但不符合性能需求的软件是不能接受的。性能测试用来测试软件在集成环境中的运行性能。在测试过程的任何步骤中都可以进行性能测试。即使是在单元级，也可以在执行测试时评估单个模块的性能。然而，只有当整个系统的所有成分完全集成之后，才能确定系统的真实性能。

性能测试经常与压力测试一起进行，且常需要硬件和软件工具。也就是说，以严格的方式测量资源（例如处理器周期）的利用往往是必要的。当有运行间歇或事件（例如中断）发生时，外部工具可以监测到，并可定期监测采样机的状态。通过检测系统，测试人员可以发现导致效率降低和系统故障的情形。

17.6.5 部署测试

在很多情况下，软件必须在多种平台及操作系统环境中运行。有时也将部署测试称为配置测试，即在软件将要运行其中的每一种环境中测试软件。另外，部署测试检查客户将要使用的所有安装程序及专业安装软件（例如“安装程序”），并检查用于向最终用户介绍软件的所有文档。

软件工具 | 测试计划与管理

[目标] 这些工具辅助软件团队根据所选择的测试策略制订计划，并进行测试过程的管理。

[机制] 这类工具提供测试计划、测试存储、管理与控制、需求追踪、集成、错误追踪和报告生成。项目经理用这些工具作为项目策划工具的补充；测试人员利用这些工具计划测试活动，以及在测试进行时控制信息的流动。

[代表性工具]^①

- QaTraq 测试用例管理工具由 Traq Soft-

ware (www.testmanagement.com) 开发，“鼓励以结构化方法进行测试管理”。

- QAComplete 由 SmartBear (<http://SmartBear.com/products/qa-tools/test-management>) 开发，为管理敏捷测试过程的各个阶段提供单点控制。
- TestWorks 由 Software Research, Inc. (<http://www.testworks.com/>) 开发，包含一个完整的、集成的成套测试工具，包括测试管理和报告工具。
- OpensourceTesting.org (www.opensou-

① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

rcetesting.org/testmgt.php) 列出了各种
开源测试管理和计划工具。

- OpensourceManagement.com (http://

www.opensourcetestmanagement.com/) 列出了各种开源测试管理和计划工具。

17.7 调试技巧

软件测试是一种能够系统地加以计划和说明的过程，可以进行测试用例设计，定义测试策略，根据预期的结果评估测试结果。

调试 (debugging) 出现在成功的测试之后。也就是说，当测试用例发现错误时，调试是使错误消除的过程。尽管调试可以是也应该是一个有序的过程，但它仍然需要很多技巧。当评估测试结果时，软件工程师经常面对的是软件问题表现出的“症状”，即错误的外部表现与其内在原因之间可能并没有明显的关系。调试就是探究这一关系的智力过程。

17.7.1 调试过程

调试并不是测试，但总是发生在测试之后^①。参看图 17-7，执行测试用例，对测试结果进行评估，而且期望的表现与实际表现不一致时，调试过程就开始了。在很多情况下，这种不一致的数据是隐藏在背后的某种原因所表现出来的症状。调试试图找到隐藏在症状背后的原因，从而使错误得到修正。

引述 一旦我们开始编程，就会惊奇地发现，程序并不是像我们想象的那样容易正确。不得不去发现错误。我能记起那一刻，意识到从那时起我将花费大部分精力去查找自己程序中的错误。

Maurice Wilkes,
discovers
debugging, 1949

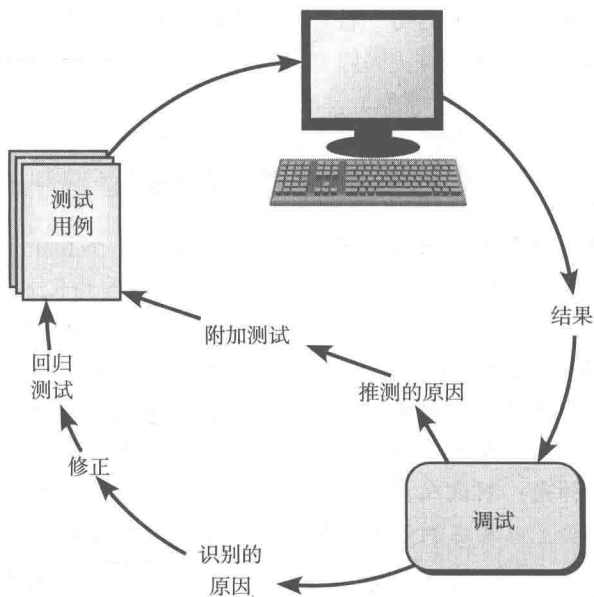


图 17-7 调试过程

① 为了对此进行说明，我们采用最广泛的可能测试视图。在软件发布之前，不仅开发者要测试软件，客户/用户在每次使用软件之前也要对其进行测试。

调试过程通常得到以下两种结果之一：(1) 发现问题原因并将其改正；(2) 未能找到问题的原因。在后一种情况下，调试人员可以假设一个原因，设计一个或多个测试用例来帮助验证这个假设，重复此过程直到改正错误。

为什么调试如此困难？在很大程度上，人类心理（17.7.2 节）与这个问题的答案间的关系比软件技术更密切。然而，软件 bug 的以下特征为我们提供了一些线索。

1. 症状与原因出现的地方可能相隔很远。也就是说，症状可能在程序的一个地方出现，而原因实际上可能在很远的另一个地方。高度耦合的构件（第 11 章）加剧了这种情况的发生；
2. 症状可能在另一个错误被改正时（暂时）消失；
3. 症状实际上可能是由非错误因素（例如舍入误差）引起的；
4. 症状可能是由不易追踪的人为错误引起的；
5. 症状可能是由计时问题而不是处理问题引起的；
6. 重新产生完全一样的输入条件是困难的（如输入顺序不确定的实时应用）；
7. 症状可能时有时无，这在软硬件耦合的嵌入式系统中尤为常见；
8. 症状可能是由分布运行在不同处理器上的很多任务引起的。

在调试过程中，我们遇到错误的范围从恼人的小错误（如不正确的输出格式）到灾难性故障（如系统失效，造成严重的经济或物质损失）。错误越严重，查找错误原因的压力也就越大。通常情况下，这种压力会使软件开发人员在修改一个错误的同时引入两个甚至更多的错误。

17.7.2 心理因素

遗憾的是，有证据表明，调试本领属于一种个人天赋。一些人精于此道，而另一些人则不然。尽管有关调试的实验证据可以有多种解释，但对于具有相同教育和经验背景的程序员来说，他们的调试能力是有很大差别的。尽管学会调试可能比较困难，但仍然可以提出一些解决问题的方法。这些方法将在下一节讨论。

提问 为什么调试如此困难？

引述 每个人都
知道调试的难度
是首次写程序的
两倍。因此，如
果你像写它时一
样聪明，那么将
如何对它进行调
试呢？

Brian Kernighan

建议 设置一个
时限，比如两个
小时。在这个时
间限制内，尽力
独自调试程序。
然后，求助。

SafeHome 调试

[场景] Ed 的工作间，进行编码和单元测试。

[人物] Ed 和 Shakira，SafeHome 软件工程团队的成员。

[对话]

Shakira (经过工作间门口时向里张望)：嘿，午饭时你在哪儿？

Ed：就在这里……工作。

Shakira：你看上去很沮丧，怎么回事？

Ed (轻声地叹息)：我一直忙于解决这个

bug，从今天早晨 9:30 发现它之后，现在已下午 2:40 了，我还没有线索。

Shakira：我想大家都同意在调试我们自己的东西时花费的时间不应该超过一小时，我们请求帮助，怎么样？

Ed：好，但是……

Shakira (走进工作间)：什么问题？

Ed：很复杂。而且，我查看这个问题已有 5 个小时，你不可能在 5 分钟内找到原因。

Shakira：真让我兴奋，什么问题？

(Ed 向 Shakira 解释问题, Shakira 看了大约 30 秒没有说什么, 然后……)

Shakira (笑了): 哦, 就是那个地方, 在循环开始之前, 变量 setAlarmCondition

是不是不应该设置为 “false”?

(Ed 不相信地盯着屏幕, 向前躬着腰, 开始对着监视器轻轻地敲自己的头。Shakira 开怀大笑, 起身走出去。)

17.7.3 调试策略

不论使用什么方法, 调试有一个基本目标: 查找造成软件错误或缺陷的原因并改正。通过系统评估、直觉和运气相结合可以实现这个目标。

总的来说, 有三种调试方法 [Mye79]: 蛮干法、回溯法及原因排除法。这三种调试方法都可以手工执行, 但现代的调试工具可以使调试过程更有效。

调试方法。蛮干法可能是查找软件错误原因最常用但最低效的方法。在所有其他方法都失败的情况下, 我们才使用这种方法。利用“让计算机自己找错误”的思想, 进行内存转储, 实施运行时跟踪, 以及在程序中添加一些输出语句。希望在所产生的大量信息里可以让我们找到错误原因的线索。尽管产生的大量信息可能最终带来成功, 但更多的情况下, 这样做只是浪费精力和时间, 它将率先耗尽我们的想法!

回溯法是比较常用的调试方法, 可以成功地应用于小程序中。从发现症状的地方开始, 向后追踪(手工)源代码, 直到发现错误的原因。遗憾的是, 随着源代码行数的增加, 潜在的回溯路径的数量可能会变得难以控制。

第三种调试方法——原因排除法——是通过演绎或归纳并引入二分法的概念来实现。对与错误出现相关的数据加以组织, 以分离出潜在的错误原因。假设一个错误原因, 利用前面提到的数据证明或反对这个假设。或者, 先列出所有可能的错误原因, 再执行测试逐个进行排除。若最初的测试显示出某个原因假设可能成立的话, 则要对数据进行细化以定位错误。

自动调试。以上调试方法都可以使用辅助调试工具。在尝试调试策略时, 调试工具为软件工程师提供半自动化的支持。Hailpern 与 Santhanam [Hai02] 总结这些工具的状况时写道: “人们已提出许多新的调试方法, 而且许多商业调试环境也已经具备。集成开发环境 (IDE) 提供了一种方法, 无需编译就可以捕捉特定语言的预置错误 (例如, 语句结束符的丢失、变量未定义等)。”可用的工具包括各种调试编译器、动态调试辅助工具 (跟踪工具)、测试用例自动生成器和交互引用映射工具。然而, 工具不能替代基于完整设计模型和清晰源代码的仔细评估。

引述 修改一个已坏的程序时, 第一步是让它重复失败 (尽可能是在最简单的例子上)。

T. Duff

软件工具 | 调试

[目标] 这些工具为那些调试软件问题的人提供自动化的帮助, 目的是洞察那些用手工调试可能难以捕捉的问题。

[机制] 大多数调试工具是针对特定编程语

言和环境的。

[代表性工具][⊖]

- Borland Silkt。由 Borland (<http://www.borland.com/products/>) 开发, 辅助测试

⊖ 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在大多数情况下, 工具名称被各自的开发者注册为商标。

和调试。

- Coverty Development Testing Platform。由 Coverty (<http://www.coverty.com/products/>) 开发, 该工具将质量测试和安全性测试引入早期开发过程。
- C++Test。由 Parasoft (www.parasoft.com) 开发, 是一个单元测试工具, 对 C 和 C++ 代码的测试提供完全的支持。调试功能有助于已发现错误的诊断。
- CodeMedic。由 NewPlanet Software (www.newplanetsoftware.com/medic/) 开发, 为标准的 Unix 调试器 gdb 提供图形界面, 且实现了它的最重要特征。gdb 目前支持 C/C++、Java、PalmOS、各种嵌入式操作系统、汇编语言、FORTRAN 和 Modula-2。
- GNATS。一个免费应用软件 (www.gnu.org/software/gnats/), 是一组用于追踪 bug 报告的工具。

GNATS。一个免费应用软件 (www.gnu.org/software/gnats/), 是一组用于追踪 bug 报告的工具。

人为因素。若不提到强有力的助手——其他人, 那么有关调试方法和调试工具的任何讨论都是不完整的。有一个新颖的观点: 每个人都可能有为某个错误一直头痛的经历[⊖]。因此, 调试的最终箴言应该是: “若所有方法都失败了, 就该寻求帮助!”

17.7.4 纠正错误

一旦找到错误, 就必须纠正。但是, 我们已提到过, 修改一个错误可能会引入其他错误, 因此, 不当修改造成的危害会超过带来的益处。Van Vleck[Van89] 提出, 在进行消除错误原因的“修改”之前, 每个软件工程师应该问以下三个问题:

1. 这个错误的原因在程序的另一部分也产生过吗? 在多数情况下, 程序的错误是由错误的逻辑模式引起的, 这种逻辑模式可能会在别的地方出现。仔细考虑这种逻辑模式可能有助于发现其他错误。
2. 进行修改可能引发的“下一个错误”是什么? 在改正错误之前, 应该仔细考虑源代码(最好包括设计)以评估逻辑与数据结构之间的耦合。若要修改高度耦合的程序段, 则应格外小心。
3. 为避免这个错误, 我们首先应当做什么呢? 这个问题是建立统计软件质量保证方法的第一步(第 16 章)。若我们不仅修改了过程, 还修改了产品, 则不仅可以排除现在的程序错误, 还可以避免程序今后可能出现的错误。

引述 最好的测试人员不是发现错误最多的人, 而是纠正错误最多的人。

Cem Kaner et al.

习题与思考题

- 17.1 用自己的话描述验证与确认的区别。两者都要使用测试用例设计方法和测试策略吗?
- 17.2 列出一些可能与独立测试组 (ITG) 的创建相关的问题。ITG 与 SQA 小组由相同的人员组成吗?
- 17.3 使用 17.1.3 节中描述的测试步骤来建立测试软件的策略总是可能的吗? 对于嵌入式系统, 会出现哪些可能的复杂情况?
- 17.4 为什么具有较高耦合度的模块难以进行单元测试?
- 17.5 “防错法” (antibugging, 17.3.1 节) 的概念是一个非常有效的方法。当发现错误时, 它提供了内置调试帮助:

⊖ 在设计软件和编码的过程中, 结对编程 (第 5 章中所讨论的极限编程模型的一部分) 提供了一种“排错”机制。

- a. 为防错法开发一组指导原则。
- b. 讨论利用这种技术的优点。
- c. 讨论利用这种技术的缺点。

- 17.6 项目的进度安排是如何影响集成测试的？
- 17.7 在所有情况下，单元测试都是可能的或是值得做的吗？提供实例来说明你的理由。
- 17.8 谁应该完成确认测试——是软件开发人员还是软件使用者？说明你的理由。
- 17.9 为本书讨论的 SafeHome 系统开发一个完整的测试策略，并以测试规格说明的方式形成文档。
- 17.10 作为一个班级项目，为你的安装开发调试指南。这个指南应该提供面向语言和面向系统的建议。这些建议是通过总结学校学习过程中所遇到的挫折得到的。从一个经过全班和老师评审过的大纲开始，并在你的局部范围内将这个指南发布给其他人。

扩展阅读与信息资源

实际上，每本软件测试的书都讨论测试策略和测试用例设计方法。Everett 和 Raymond (《Software Testing》，Wiley-IEEE Computer Society Press, 2007)、Black (《Pragmatic Software Testing》，Wiley, 2007)、Spiller 和他的同事 (《Software Testing Process: Test Management》，Rocky Nook, 2007)、Perry (《Effective methods for Software Testing》，3rd ed., Wiley, 2005)、Lewis (《Software Testing and Continuous Quality Improvement》，2nd ed., Auerbach, 2004)、Loveland 和他的同事 (《Software Testing Techniques》，Charles River Media, 2004)、Burnstein (《Practical Software Testing Techniques》，Springer, 2003)、Dustin (《Effective Software Testing》，Addison-Wesley, 2002) 以及 Kaner 和他的同事 (《Lessons learned in Software Testing》，Wiley, 2001) 所写的书只是讨论测试原理、概念、策略和方法的众多书籍中的一小部分。

对于敏捷软件开发方法有兴趣的读者，Gartner (《ATDD by Example: A Practical Guide to Acceptance Test-Driven Development》，Addison-Wesley, 2012)、Crispin 和 Gregory (《Agile Testing: A Practical Guide for Testers and Teams》，Addison-Wesley, 2009)、Crispin 和 House (《Testing Extreme Programming》，Addison-Wesley, 2002) 以及 Beck (《Test Driven Development: By Example》，Addison-Wesley, 2002) 针对极限编程技术描述了测试策略与战术。Kaner 和他的同事 (《Lessons Learned in Software Testing》，Wiley, 2001) 描述了每个测试人员应该学习的 300 多条实用的“教训”(指导原则)。Watkins (《Testing IT: An Off-the Shelf Testing Process》(2nd ed.), Cambridge University Press, 2010) 为所有类型的软件(开发的和获取的)建立了有效的测试框架。Manages 和 O'Brien (《Agile Testing with Ruby and Rails》，Apress, 2008) 描述了针对 Ruby 编程语言和 Web 框架的测试策略和技术。

Bashir 和 Goel (《Testing Object-Oriented Software》，Springer-Verlag, 2012)、Sykes 和 McGregor (《Practical Guide to Testing Object-Oriented Software》，Addison-Wesley, 2001)、Binder (《Testing Object-Oriented Systems》，Addison-Wesley, 1999)、Kung 和他的同事 (《Testing Object-Oriented Software》，IEEE Computer Society Press, 1998) 以及 Marick (《The Craft of Software Testing》，Prentice-Hall, 1997) 描述了测试面向对象系统的策略与方法。

Grotker 和他的同事 (《The Developer's Guide to Debugging》(2nd ed.), CreateSpace Independent Publishing, 2012)、Whittaker (《Exploratory Testing》，Addison-Wesley, 2009)、Zeller (《Why Programs Fail: A Guide to Systematic Debugging》(2nd ed.), Morgan Kaufmann, 2009)、Butcher (《Debug It!》，Pragmatic Bookshelf, 2009)、Agans (《Debugging》，Amacon, 2006) 以及 Tells 和 Heieh (《The Science of Debugging》，The Coreolis Group, 2001) 所编写的书中包括调试指南。Kaspersky (《Hacker Debugging Uncovered》，A-list Publishing, 2005) 讲述了调试工具的技术。Younessi (《Object-Oriented Defect Manag-

ement of Software》, Prentice-Hall, 2002) 描述了面向对象系统的缺陷管理技术。Beizer[Bei84] 描述了有趣的 “bug 分类”, 这种分类引领了很多制定测试计划的有效方法。

Graham 和 Fewster (《 Experience of Test Automation 》, Addison-Wesley, 2012) 以及 Dustin 和他的同事 (《 Implementing Automated Software Testing 》, Addison-Wesley, 2009) 所编写的书讨论了自动测试。Hunt 和 John (《 Java Performance 》, Addison-Wesley, 2011)、Hewardt 和他的同事 (《 Advanced .NET Debugging 》, Addison-Wesley, 2009)、Matloff 和他的同事 (《 The Art of Debugging with GDB, DDD, and Eclipse 》, No Starch Press, 2008)、Madisetti 和 Akgul (《 Debugging Embedded Systems 》, Springer, 2007)、Robbins (《 Debugging Microsoft .NET 2.0 Applications 》, Microsoft Press, 2005)、Best (《 Linux Debugging and Performance Tuning 》, Prentice Hall, 2005)、Ford 和 Teorey (《 Practical Debugging in C++ 》, Prentice Hall, 2002) 以及 Brown (《 Debugging Perl 》, McGraw-Hill, 2000)、Mitchell (《 Debugging Java 》, McGraw-Hill, 2000) 都针对书名所指的环境, 讲述了调试的特殊性质。

从网上可以获得大量有关软件测试策略的信息资源。与软件测试策略有关的最新的参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

测试传统的应用软件

要点浏览

概念: 一旦生成了源代码,就必须对软件进行测试,以便在交付给客户之前尽可能多地发现(和改正)错误。我们的目标是设计一系列极有可能发现错误的测试用例。但是,如何做呢?这就是软件测试技术发挥作用的地方。这些技术为设计测试提供系统化的指导:(1) 执行每个软件构件的内部逻辑和接口;(2) 测试程序的输入和输出域以发现程序功能、行为和性能方面的错误。

人员: 在测试的早期阶段,软件工程师完成所有的测试。然而,随着测试过程的进行,测试专家可能介入。

重要性: 评审及其他软件质量保证活动可以且确实能够发现错误,但只有这些做法是远远不够的。每次执行程序时,用户都在测试它。因此,在程序交付给客户之前,就必须以发现并消除错误为目的来执行它。为了尽可能多地发现错误,

必须系统化地执行测试,而且必须利用严格的技术来设计测试用例。

步骤: 对于传统的应用软件,可从两个不同的视角测试软件:(1) 利用“白盒”测试用例设计技术执行程序内部逻辑;(2) 利用“黑盒”测试用例设计技术确认软件需求。用例可辅助测试的设计,在软件确认的层面发现错误。在每种情况下,其基本意图都是以最少的工作量和最少的时间来发现最大数量的错误。

工作产品: 设计一组测试用例使其不仅测试内部逻辑、接口、构件协作,还测试外部需求,并形成文档。定义期望结果,并记录实际结果。

质量保证措施: 当开始测试时,改变视角,努力去“破坏”软件!规范化地设计测试用例,并对测试用例进行周密的评审。另外,评估测试覆盖率并追踪错误检测活动。

对于本质上具有建设性的软件工程师来说,测试展示出的是有趣的异常现象。测试要求开发者首先抛弃“刚开发的软件是正确的”这一先入为主的观念,然后努力去构造测试用例来“破坏”软件。Beizer[Bei90] 有效地描述了这种情况:

有这样一个神话:若我们确实擅长编程,就应当不会有错误。只要 we 确实很专注,只要每个人都使用结构化编程,采用自顶向下的设计方法……那么就不应该有错误。所以才有了这样的神话。神话中讲道:由于我们并不擅长所做的事,因此有错误存在。若不擅长,就应当感到内疚。因此,测试和测试用例的设计是对失败的承认,也是失败的一剂良药。测试的枯燥是对我们犯下的错误的处罚。为什么被罚?由于我们是人类?为

关键概念

基本路径测试

黑盒测试

边界值分析

控制结构测试

环路复杂性

等价类划分

流程图

方法

基于模型的测试

白盒测试

什么内疚？由于没能达到非人的完美境界？由于没能区分另一个程序员所想的和所说的之间存在的差异？由于没有心灵感应？由于没有解决交流问题？……由于人类四千年历史的缘故？

测试应该灌输内疚感吗？测试真的是摧毁性的吗？这些问题的回答是“不”！

本章针对传统的应用软件讨论软件测试用例设计技术。测试用例设计关注创建测试用例的一系列技术，这些测试用例的设计符合总体测试目标及第 17 章所述的测试策略。

18.1 软件测试基础

测试的目标是发现错误，并且好的测试发现错误的可能性较大。因此，软件工程师在设计与实现基于计算机的系统或产品时，应该想着可测试性。同时，测试本身必须展示一系列特征，达到以最少工作量发现最多错误的目标。

可测试性。James Bach^①为可测试性提供了下述定义：“软件可测试性就是（计算机程序）能够被测试的容易程度。”可测试的软件应具有下述特征。

可操作性。“运行得越好，越能有效地测试。”若设计和实现系统时具有质量意识，那么妨碍测试执行的错误将很少，从而使测试顺利进行。

可观察性。“你所看见的就是你所测试的。”作为测试的一部分所提供的输入会产生清楚的输出。测试执行期间系统状态和变量是可见的或可查询的，不正确的输出易于识别，内部错误会被自动检测和报告，源代码是可访问的。

可控制性。“对软件控制得越好，测试越能被自动执行和优化。”通过输入的某些组合可以产生所有可能的输出，并且输入/输出格式是一致的和结构化的。通过输入的组合，所有代码都可以执行到。测试工程师能够控制软硬件的状态和变量，能够方便地对测试进行说明、自动化执行和再现。

可分解性。“通过控制测试范围，能够更快地孤立问题，完成更灵巧的再测试。”软件由能够进行单独测试的独立模块组成。

简单性。“需要测试的内容越少，测试的速度越快。”程序应该展示功能简单性（例如，程序特性集是满足需求的最低要求）、结构简单性（例如，将体系结构模块化以限制错误的传播）以及代码简单性（例如，采用编码标准以使代码易于审查和维护）。

稳定性。“变更越少，对测试的破坏越小。”软件的变更不经常发生，变更发生时是可以控制的，且不影响已有的测试，软件失效后得到良好恢复。

易理解性。“得到的信息越多，进行的测试越灵巧。”体系结构设计以及内部构件、外部构件和共享构件之间的依赖关系能被较好地理解。技术文档可随时获取、组织合理、具体、详细且准确。设计的变更要通知测试人员。

可以使用 Batch 所建议的属性来开发易于测试的软件工作产品。

测试特征。关于测试本身有哪些特征呢？Kaner、Falk 和 Nguyen[Kan93] 提出“好”的测试具有以下属性。

好的测试具有较高的发现错误的可能性。为达到这个目标，测试人员必须理解软件并尝

引述 每个程序都做对某件事，可是那恰恰不一定是我们想让它做的事情！

作者不详

提问 可测试性的特征是什么？

引述 软件中的错误比其他技术中的错误更普通、普遍且更烦人。

David Parnas

① 后面几段取得了 James Bach (copyright 1994) 的使用许可，并对最初出现在新闻组 comp.software-eng 的资料进行了改编。

试设想软件怎样才能失败。

好的测试是不冗余的。测试时间和资源是有限的,执行与另一个测试有同样目标的测试是没有意义的。每个测试都应该有不同的目标(即使是细微的差别)。

好的测试应该是“最佳品种”[Kan93]。在一组具有类似目的的测试中,时间和资源的有限性会迫使只运行最有可能发现所有类别错误的测试。

好的测试应该既不太简单也不太复杂。尽管将一系列测试连接为一个测试用例有时是可能的,但潜在的副作用会掩盖错误。通常情况下,应该独立执行每个测试。

提问 什么才是“好”的测试?

SafeHome 设计独特的测试

[场景] Vinod 的工作间。

[人物] Vinod 与 Ed, SafeHome 软件工程团队成员。

[对话]

Vinod: 这些是你打算用于测试操作 passwordValidation 的测试用例吗?

Ed: 是的,它们应该能覆盖用户进入时所有可能输入的密码。

Vinod: 让我看看……你提到正确的密码是 8080,对吗?

Ed: 嗯。

Vinod: 你指定密码 1234 和 6789 是要测试在识别无效密码方面的错误?

Ed: 对,我也测试与正确密码相接近的密

码,如 8081 和 8180。

Vinod: 那是可行的。但是我并不认为运行 1234 和 6789 两个输入有多大意义。这两个输入是冗余的……它们在测试同样的事情,不是吗?

Vinod: 确实是这样。倘若输入 1234 不能发现错误,换句话说,操作 passwordValidation 指出它是无效密码,那么输入 6789 也不可能显示任何新的东西。

Ed: 我明白你的意思。

Vinod: 我不是吹毛求疵,只是我们做测试的时间有限,因此,好的方法是运行最有可能发现新错误的测试。

Ed: 没问题……我再想想。

18.2 测试的内部视角和外部视角

任何工程化的产品(以及大多数其他东西)都可以采用以下两种方式之一进行测试:(1)了解已设计的产品要完成的指定功能,可以执行测试以显示每个功能是可操作的,同时,查找在每个功能中的错误;(2)了解产品的内部工作情况,可以执行测试以确保“所有的齿轮吻合”——即内部操作依据规格说明执行,而且对所有的内部构件已进行了充分测试。第一种测试方法采用外部视角,也称为黑盒测试;第二种方法采用内部视角,也称为白盒测试。^①

黑盒测试暗指在软件接口处执行测试。黑盒测试检查系统的功能方面,而不考虑软件的内部结构。软件的白盒测试是基于过程细节的封闭检查。通过提供检查特定条件集或循环的测试用例,测试将贯穿软件的逻辑路径和构件间的协作。

乍一看,好像是全面的白盒测试将获得“100% 正确的程序”。我们需要做的只是识别所有的逻辑路径,开发相应的测试用例、例执行测试用例并

引述 在设计测试用例中只有一条规则,那就是覆盖所有特征,但并不创建太多的测试用例。

Tsuneo Yamaura

关键点 只有在构件级设计(或源代码)存在之后,才设计白盒测试。此时,一定要获得程序的逻辑细节。

① 术语功能测试和结构测试有时分别用于代替黑盒测试和白盒测试。

评估结果；即生成测试用例，彻底地测试程序逻辑。遗憾的是，穷举测试存在某种逻辑问题，即使对于小程序，可能的逻辑路径的数量也可能非常大。然而，不应该觉得白盒测试不切实际而抛弃这种方法。可以选择并测试有限数量的重要逻辑路径，检测重要数据结构的有效性。

信息栏 穷举测试

考虑 100 行的 C 语言程序。一些基本的数据声明之后，程序包含两个嵌套循环，依靠输入指定的条件，每个从 1 到 20 次进行循环。在内部循环中，需要 4 个 if-then-else 结构。这个程序中大约有 1014 个可能的执行路径！

为了说明这个数字代表的含义，我们假设已经开发了一个神奇的测试处理器（“神奇”意味着没有这样的处理器存在）

来做穷举测试。在 1 毫秒内，处理器可以开发一个测试用例、执行测试用例并评估测试结果。若处理器每天工作 24 小时，每年工作 365 天，要对这个程序做完穷举测试，需要工作 3170 年。不可否认，这将对大多数的开发进度造成巨大障碍。

因此，可以肯定地说，对于大型软件系统，穷举测试是不可能的。

18.3 白盒测试

白盒测试有时也称为玻璃盒测试或结构化测试，是一种测试用例设计方法，它利用作为构件级设计的一部分所描述的控制结构来生成测试用例。利用白盒测试方法导出的测试用例可以：（1）保证一个模块中的所有独立路径至少被执行一次；（2）对所有的逻辑判定均需测试取真（true）和取假（false）两个方面；（3）在上下边界及可操作的范围内执行所有循环；（4）检验内部数据结构以确保其有效性。

引述 bug 潜伏在角落并在边界处聚集。

Boris Beizer

18.4 基本路径测试

基本路径测试是由 Tom McCabe[McC76] 首先提出的一种白盒测试技术。基本路径测试方法允许测试用例设计者计算出过程设计的逻辑复杂性测量，并以这种测量为指导来定义执行路径的基本集。执行该基本集导出的测试用例保证程序中的每一条语句至少执行一次。

18.4.1 流图表示

在介绍基本路径方法之前，必须介绍一种简单的控制流表示方法，称为流图（或程序图）^①。流图利用图 18-1 所示的表示描述逻辑控制流。每种结构化构造（第 13 章）都有相应的流图符号。

为了说明流图的使用，考虑图 18-2a 所示的过程设计表示。这里，流程图用于描述程序的控制结构。图 18-2b 将这个流程图映射为相应的流图（假设流程图的菱形判定框中不包含复合条件）。在图 18-2b 中，圆称为流图结

建议 仅在构件的逻辑结构复杂的情况下，才应该画流图。使用流图可以更轻易地追踪程序路径。

^① 事实上，不使用流图也可以执行基本路径测试方法，但是，流图是用于理解控制流和解释方法的一种有用表示。

18.4.2 独立程序路径

独立路径是任何贯穿程序的、至少引入一组新处理语句或一个新条件的路径。当按照流程图进行描述时，独立路径必须沿着至少一条边移动。这条边在定义该路径之前未被遍历。例如，图 18-2b 所示流图的一组独立路径如下：

路径 1: 1-11

路径 2: 1-2-3-4-5-10-1-11

路径 3: 1-2-3-6-8-9-10-1-11

路径 4: 1-2-3-6-7-9-10-1-11

注意，每条新的路径引入一条新边，路径

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

不是一条独立路径，因为它不过是已提到路径的简单连接，而没有引入任何新边。

路径 1、2、3 和 4 构成图 18-2b 所示流图的基本集合。也就是说，若设计测试以强迫执行这些路径（基本集合），则可以保证程序中的每条语句至少执行一次，且每个条件的取真和取假都被执行。应该注意到，基本集合不是唯一的。事实上，对给定的过程设计，可以导出很多不同的基本集合。

如何知道要找出多少路径？环复杂性的计算提供了答案。环复杂性是一种软件度量，它为程序的逻辑复杂度提供了一个量化的测度。用在基本路径测试方法的环境下时，环复杂性的值定义了程序基本集合中的独立路径数，并提供了保证所有语句至少执行一次所需测试数量的上限。

环复杂性以图论为基础，并提供了非常有用的软件度量。可以通过以下三种方法之一来计算环复杂性。

1. 流图中域的数量与环复杂性相对应。
2. 对于流图 G ，环复杂性 $V(G)$ 定义如下：

$$V(G) = E - N + 2$$

其中 E 为流图的边数， N 为流图的结点数。

3. 对于流图 G ，环复杂性 $V(G)$ 也可以定义如下：

$$V(G) = P + 1$$

其中 P 为包含在流图 G 中的判定结点数。

再回到图 18-2b 中的流图，环复杂性可以通过上述三种算法来计算。

1. 该流图有 4 个域。
2. $V(G) = 11$ （边数） $- 9$ （结点数） $+ 2 = 4$ 。
3. $V(G) = 3$ （判定结点数） $+ 1 = 4$ 。

因此，图 18-2b 中流图的环复杂性是 4。

更重要的是， $V(G)$ 的值提供了组成基本集合的独立路径的上界，并由此得出覆盖所有程序语句所需设计和运行的测试数量的上界。

建议 在预见易于出错的模块方面，环复杂性是一种有用的度量，可以用于做测试计划以及测试用例设计。

提问 如何计算环复杂性？

关键点 环复杂性提供保证程序中每条语句至少执行一次所需测试用例数的上界。

工程团队成员，他们正在为安全功能准备测试计划。

[对话]

Shakira：看，我知道应该对安全功能的所有构件进行单元测试，但是，如果考虑所有必须测试的操作的数量，工作量就太大了，我不知道……可能我们应该放弃白盒测试，将所有的构件集成在一起，开始执行黑盒测试。

Vinod：你估计我们没有足够的时间做构件测试、检查操作，然后集成，是不是？

Shakira：第一次增量测试的最后期限离我们很近了……是的，我有点担心。

Vinod：你为什么不对最有可能出错的操作执行白盒测试呢？

Shakira (愤怒地)：我怎么能够准确地知道哪个是最易出错的呢？

Vinod：环复杂性。

Shakira：嗯？

Vinod：环复杂性。只要计算每个构件中每个操作的环复杂性。看看哪些操作的 $V(G)$ 具有最高值。那些操作就是最有可能出错的操作。

Shakira：怎么计算 $V(G)$ 呢？

Vinod：那相当容易。这里有本书说明了怎么计算。

Shakira (翻看那几页)：好了，这计算看上去并不难。我试一试。具有最高 $V(G)$ 值的就是要做白盒测试的候选操作。

Vinod：但还要记住，这并不是绝对的，那些 $V(G)$ 值低的构件还是可能有错的。

Shakira：好吧。但这至少降低了必须进行白盒测试的构件数。

18.4.3 生成测试用例

基本路径测试方法可以应用于过程设计或源代码。在本节中，我们将基本路径测试描述为一系列步骤。以图 18-4 中用 PDL 描述的过程 average 为例，说明测试用例设计方法中的各个步骤。注意，尽管过程 average 是一个非常简单的算法，但却包含了复合条件与循环。下列步骤可用于生成基本测试用例集。

1. 以设计或源代码为基础，画出相应的流程图。利用 18.4.1 节给出的符号和构造规则创建流程图。参见图 18-4 中过程 average 的 PDL 描述，将那些 PDL 语句进行编号，并映射到相应的流程图结点，以此来创建流程图。图 18-5 给出了相应的流程图。
2. 确定所得流程图的环复杂性。通过运用 18.4.2 节描述的算法来确定环复杂性 $V(G)$ 的值。应该注意到，不建立流程图也可以确定 $V(G)$ ，方法是通过计算 PDL 中条件语句的数量（过程 average 的复合条件语句计数为 2），然后加 1。在图 18-5 中：

$$V(G) = 6(\text{域数})$$

$$V(G) = 17(\text{边数}) - 13(\text{结点数}) + 2 = 6$$

$$V(G) = 5(\text{判定结点数}) + 1 = 6$$

3. 确定线性独立路径的基本集合。 $V(G)$ 的值提供了程序控制结构中线性独立路径的数量。在过程 average 中，我们指定了 6 条路径：

路径 1: 1-2-10-11-13

路径 2: 1-2-10-12-13

引述 犯错误的是人，发现错误的是神。

Robert Dunn

引述 只是由于在将 64 位浮点值转换为 16 位整数的操作中包含了一个软件缺陷（代码错误），Ariane 5 型火箭在升空时发生爆炸。这枚火箭和它的 4 颗卫星都没有投保，它们价值 5 亿美元。如果进行路径测试是可以发现这个错误的，但由于预算原因被否决了。

一篇新闻报道

- 路径 3: 1-2-3-10-11-13
- 路径 4: 1-2-3-4-5-8-9-2-...
- 路径 5: 1-2-3-4-5-6-8-9-2-...
- 路径 6: 1-2-3-4-5-6-7-8-9-2-...

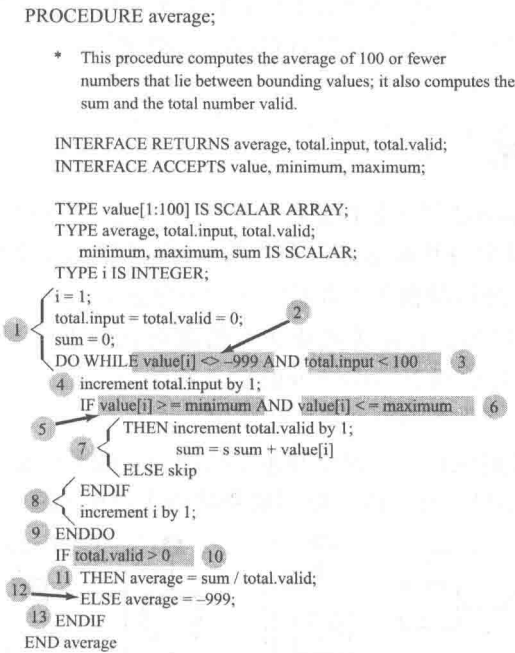


图 18-4 已标识结点的 PDL

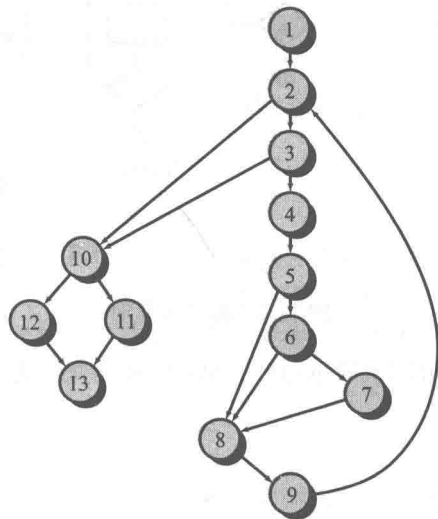


图 18-5 过程 average 的流图

路径 4、5、6 后面的省略号 (...) 表示可加上控制结构其余部分的任意路径。在设计测试用例的过程中，经常通过识别判定结点作为导出测试用例的辅助手段。本例中，结点 2、3、5、6 和 10 为判定结点。

4. 准备测试用例，强制执行基本集合中的每条路径。测试人员应该选择测试数据，以便在测试每条路径时适当地设置判定结点的条件。执行每个测试用例并将结果与期望值进行比较。一旦完成了所有的测试用例，测试人员就可以确信程序中所有的语句至少已被执行一次。

注意，某些独立路径（本例中的路径1）不能单独进行测试。也就是说，遍历路径所需的数据组合不能形成程序的正常流。在这种情况下，将这些路径作为另一个路径的一部分进行测试。

18.5 控制结构测试

18.4 节所描述的基本路径测试是控制结构测试技术之一。虽然基本路径测试简单且高效，但其本身并不充分。本节简单讨论控制结构测试的其他变体，这些技术拓宽了测试的覆盖率并提高了白盒测试的质量。

条件测试 [Tai89] 通过检查程序模块中包含的逻辑条件进行测试用例设计。数据流测试 [Fra93] 根据程序中变量的定义和使用位置来选择程序的测试路径。

循环测试是一种白盒测试技术，完全侧重于循环构建的有效性。可以定义 4 种不同的循环 [Bei90]：简单循环、串接循环、嵌套循环和非结构化循环（图 18-6）。

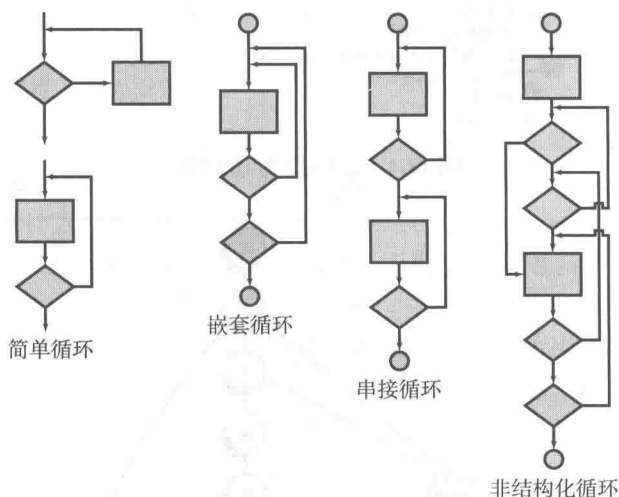


图 18-6 循环的类别

简单循环。下列测试集可用于简单循环，其中， n 是允许通过循环的最大次数。

1. 跳过整个循环。
2. 只有一次通过循环。
3. 两次通过循环。
4. m 次通过循环，其中 $m < n$ 。
5. $n-1$ 、 n 、 $n+1$ 次通过循环。

嵌套循环。若将简单循环的测试方法扩展应用于嵌套循环，则可能的测试数将随着嵌套层次的增加而成几何级数增长。这将导致不切实际的测试数量。Beizer[Bei90] 提出了一种有助于减少测试数的方法。

引述 将执行测试看得比设计测试更重要是一个典型的错误。

Brian Marick

引述 优秀的测试人员是注意到“奇怪的事情”就会对它采取行动的大师。

Brian Marick

1. 从最内层循环开始, 将其他循环设置为最小值。
2. 对最内层循环执行简单循环测试, 而使外层循环的迭代参数 (例如循环计数) 值最小, 并对范围以外或不包括在内的值增加其他测试。
3. 由内向外构造下一个循环的测试, 但使其他外层循环具有最小值, 并使其他嵌套循环为“典型”值。
4. 继续上述过程, 直到测试完所有的循环。

串接循环。若串接循环的每个循环彼此独立, 则可以使用简单循环测试方法。然而, 若两个循环串接起来, 且第一个循环的循环计数为第二个循环的初始值, 则这两个循环并不独立。若循环不独立, 则建议使用嵌套循环的测试方法。

非结构化循环。若有可能, 应该重新设计这类循环以反映结构化程序结构的使用 (第13章)。

建议 不能对非结构化循环进行有效测试, 需要对它们进行重新设计。

18.6 黑盒测试

黑盒测试也称行为测试或功能测试, 侧重于软件的功能需求。黑盒测试使软件工程师能设计出可以测试程序所有功能需求的输入条件集。黑盒测试并不是白盒测试的替代品, 而是作为发现其他类型错误的辅助方法。

黑盒测试试图发现以下类型的错误: (1) 不正确或遗漏的功能; (2) 接口错误; (3) 数据结构或外部数据库访问错误; (4) 行为或性能错误; (5) 初始化和终止错误。

与白盒测试不同, 白盒测试在测试过程的早期执行, 而黑盒测试倾向于应用在测试的后期阶段 (第17章)。黑盒测试故意不考虑控制结构, 而是侧重于信息域。设计黑盒测试要回答下述问题:

- 如何测试功能的有效性?
- 如何测试系统的行为和性能?
- 哪种类型的输入会产生好的测试用例?
- 系统是否对特定的输入值特别敏感?
- 如何分离数据类的边界?
- 系统能承受什么样的数据速率和数据量?
- 特定类型的数据组合会对系统运行产生什么样的影响?

提问 什么是黑盒测试必须回答的问题?

通过运用黑盒测试技术, 可以生成满足下述准则的测试用例集 [Mye79]: 能够减少达到合理测试所需的附加测试用例数, 并且能够告知某些错误类型是否存在, 而不是仅仅知道与特定测试相关的错误。

18.6.1 等价类划分

等价类划分是一种黑盒测试方法, 它将程序的输入划分为若干个数据类, 从中生成测试用例。理想的测试用例可以单独发现一类错误 (例如, 所有字符数据处理不正确), 否则在观察到一般的错误之前需要运行许多测试用例。

等价类划分的测试用例设计是基于对输入条件的等价类进行评估。利用上节引入的概念, 若对象可以由具有对称性、传递性和自反性的关系连接, 则存在等价类 [Bei95]。等价类表示输入条件的一组有效的或无效的状态。通常情况下, 输入条件要么是一个特定值、一

个数据域、一组相关的值，要么是一个布尔条件。可以根据下述指导原则定义等价类。

提问 如何为测试定义等价类？

1. 若输入条件指定一个范围，则可以定义一个有效等价类和两个无效等价类。
2. 若输入条件需要特定的值，则可以定义一个有效等价类和两个无效等价类。
3. 若输入条件指定集合的某个元素，则可以定义一个有效等价类和一个无效等价类。
4. 若输入条件为布尔值，则可以定义一个有效等价类和一个无效等价类。

通过运用设计等价类的指导原则，可以为每个输入域数据对象设计测试用例并执行。选择测试用例以便一次测试一个等价类的尽可能多的属性。

18.6.2 边界值分析

大量错误发生在输入域的边界处，而不是发生在输入域的“中间”。这是将边界值分析（Boundary Value Analysis, BVA）作为一种测试技术的原因。边界值分析选择一组测试用例检查边界值。

引述 测试代码的一种有效方式是在其自然边界处运行它。

Brian Kernighan

边界值分析是一种测试用例设计技术，是对“等价划分”的补充。BVA 不是选择等价类的任何元素，而是在等价类“边缘”上选择测试用例。BVA 不是仅仅侧重于输入条件，它也从输出域中导出测试用例 [Mye79]。

关键点 通过侧重考虑一个等价类“边界”处的数据，BVA 扩展了等价类划分。

BVA 的指导原则在很多方面类似于等价划分的原则。

1. 若输入条件指定为以 a 和 b 为边界的范围，则测试用例应该包括 a 和 b ，略大于和略小于 a 和 b 。
2. 若输入条件指定为一组值，则测试用例应当执行其中的最大值和最小值，以及略大于和略小于最大值和最小值的值。
3. 指导原则 1 和 2 也适用于输出条件。例如，工程分析程序要求输出温度和压强的对照表，应该设计测试用例创建输出报告，输出报告可生成所允许的最大（和最小）数目的表项。
4. 若内部程序数据结构有预定义的边界值（例如，表具有 100 项的定义限制），则一定要设计测试用例，在其边界处测试数据结构。

大多数软件工程师会在某种程度上凭直觉完成 BVA。通过运用这些指导原则，边界测试会更加完全，从而更有可能发现错误。

18.7 基于模型的测试

基于模型的测试（Model-based Testing, MBT）是一种黑盒测试技术，它使用需求模型中的信息作为生成测试用例的基础 [DAC03]。在很多情况下，基于模型的测试技术使用 UML 状态图——一种行为模型（第 10 章）作为测试用例设计的基础^①。MBT 技术需要以下 5 个步骤。

引述 当你在代码中寻找错误时，很难发现它；当你认为自己的代码没有错误时，就更难发现它。

Steve McConnell

1. 分析软件的已有行为模型或创建一个行为模型。回忆一下，行为模型指明软件是如何响应外部事件或刺激的。为了创建行为模型，我们需要执行第 10 章所讨论的步骤：（1）评价所有的用例，以完全

① 当软件需求是用决策表、语法或 Markov 链表示时，也可以使用基于模型的测试 [DAC03]。

理解系统内的交互顺序；(2) 标识驱动交互顺序的事件，并理解这些事件如何与特定的对象相关；(3) 为每个用例创建交互顺序；(4) 构造系统的 UML 状态图（例如，见图 10-1）；(5) 评审行为模型，验证其精确性和一致性。

2. 遍历行为模型，并标明促使软件在状态之间进行转换的输入。输入将触发事件，使转换发生。
3. 评估行为模型，并标注当软件在状态之间转换时所期望的输出。回想一下，每个转换都由一个事件触发，作为转换的结果，某些方法会被调用并产生输出。对于步骤 2 指定的每个输入（用例）集合，指定所期望的输出，以说明它们在行为模型中的特点。
4. 运行测试用例。可以手工执行测试，也可以创建测试脚本并使用测试工具执行测试。
5. 比较实际结果和期望结果，并根据需要进行调整。

MBT 可帮助我们发现软件行为中的错误，因此，它在测试事件驱动的应用时也非常有用。

习题与思考题

- 18.1 Myers[Mye79] 用以下程序作为对测试能力的自我评估：某程序读入 3 个整数值，这 3 个整数值表示三角形的 3 条边。该程序打印信息以表明三角形是不规则的、等腰的或等边的。开发一组测试用例测试该程序。
- 18.2 设计并实现习题 18.1 描述的程序（适当时使用错误处理）。从该程序中导出流程图并用基本路径测试方法设计测试，以保证程序中的所有语句都被测试到。执行测试用例并显示结果。
- 18.3 你能够想出 18.1 节中没有讨论的其他测试目标吗？
- 18.4 选择一个你最近设计和实现的构件。设计一组测试用例，保证利用基本路径测试执行所有的语句。
- 18.5 说明、设计和实现一个软件工具，使其能够对你所选的程序设计语言计算环复杂性。在你的设计中，利用图矩阵作为有效的数据结构。
- 18.6 阅读 Beizer[Bei95] 或相关的网络资源（例如，www.laynetworks.com/Discrete%20Mathematics_1g.htm），并确定如何扩展习题 18.5 所开发的程序以适应各种连接权值。扩展你的工具以处理执行概率或连接处理时间。
- 18.7 设计一个自动化测试工具，使其能够识别循环并按照 18.5 节中的方法分类。
- 18.8 扩展习题 18.7 中描述的工具，为曾经遇到的每个循环类生成测试用例。与测试人员交互地完成这个功能是有必要的。
- 18.9 至少给出 3 个例子，在这些例子中，黑盒测试能够给人“一切正常”的印象，而白盒测试可能发现错误。再至少给出 3 个例子，在这些例子中白盒测试可能给人“一切正常”的印象，而黑盒测试可能发现错误。
- 18.10 穷举测试（即便对非常小的程序）是否能够保证程序 100% 正确？
- 18.11 测试你经常使用的某个应用软件的用户手册（或帮助设施）。在文档中至少找到一个错误。

扩展阅读与信息资源

实际上，所有软件测试方面的书籍都同时考虑测试策略和测试技术。因此，第 17 章的推荐读物同样适用于本章。有许多讨论测试原理、概念、策略和方法的书籍，下面的书籍只是其中的一小部分：Burnstein（《Practical Software Testing》，Springer, 2010）、Crispin 和 Gregory（《Agile Testing: A Practical Guide for Testers and Agile Teams》，Addison-Wesley, 2009）、Lewis（《Software Testing and Continuous Quality Improvement》，3rd ed., Auerbach, 2008）、Ammann 和 Offutt（《Introduction to Software Testing》，Cambridge University Press, 2008）、Everett 和 McCleod（《Software Testing》，Wiley-IEEE Computer Society Press, 2007）、Black（《Pragmatic Software Testing》，Wiley, 2007）、Spiller 和他的同事（《Software

Testing Process : Test Management 》, Rocky Nook, 2007)、Perry (《 Effective Methods for Software Testing 》, 3rd ed., Wiley, 2006)、Loveland 和他的同事 (《 Software Testing Techniques 》, Charles River Media, 2004)、Dustin (《 Effective Software Testing 》, Addison-Wesley, 2002)、Craig 和 Kaskiel (《 Systematic Software Testing 》, Artech House, 2002)、Tamres (《 Introducing Software Testing 》, Addison-Wesley, 2002) 以及 Whittaker (《 Exploratory Software Testing: Tips, Tricks, and Techniques to Guide Test Design 》, Addison-Wesley, 2009 和 《 How to Break Software 》, Addison-Wesley, 2002)。

Myers[Mye79] 经典书籍的第 3 版 (《 The Art of Software Testing, 3rd ed., Wiley, 2011 》) 由 Myers 及其同事编写, 非常详细地讲述了测试用例的设计技术。Black (《 Managing the Testing Process 》, 3rd ed., Wiley, 2009)、Jorgensen (《 Software Testing: A Craftsman's Approach 》, 3rd ed., CRC Press, 2008)、Pezze 和 Young (《 Software Testing and Analysis 》, Wiley, 2007)、Perry (《 Effective Methods for Software Testing 》, 3rd ed., Wiley, 2006)、Copeland (《 A Practitioner's Guide to Software Test Design 》, Artech, 2003) 以及 Hutcheson (《 Software Testing Fundamentals 》, Wiley, 2003) 都提供了测试用例设计方法和技术的有用介绍。Beizer[Bei90] 的经典文本全面介绍了白盒测试技术, 引入了数学级别上的严格性, 这在其他测试方面的论述中一般是不具备的。他后来的书籍 [Bei95] 对重要方法作了简明介绍。

软件测试是一种资源密集型的活动。为此, 许多组织为部分测试过程提供了自动化支持。Graham 和她的同事 (《 Experiences of Test Automation: Case Studies of Software Test Automation 》, Addison-Wesley, 2012 和 《 Software Test Automation 》, Addison-Wesley, 1999)、Li 和 Wu (《 Effective Software Test Automation 》, Sybex, 2004)、Mosely 和 Posey (《 Just Enough Software Test Automation 》, Prentice Hall, 2002)、Poston (《 Automating Specification-Based Software Testing 》, IEEE Computer Society, 1996) 以及 Dustin、Rashka 和 Poston (《 Automated Software Testing: Introduction, Management, and Performance 》, Addison-Wesley, 1999) 讨论了自动化测试的工具、策略和方法。Nquyen 和他的同事 (《 Happy About Global Software Test Automation 》, Happy About Press, 2006) 介绍了测试自动化的执行视图。

Meszaros (《 Unit Test Patterns: Refactoring Test Code 》, Addison-Wesley, 2007)、Thomas 和他的同事 (《 Java Testing Patterns 》, Wiley, 2004) 以及 Binder[Bin99] 描述了测试模式, 包括方法测试、类 / 簇测试、子系统测试、可复用构件测试、框架测试、系统测试、测试自动化及特定数据库测试。

从网上可以获得大量的有关测试用例设计方法的信息。有关测试技术的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

测试面向对象的应用

要点浏览

概念: 面向对象 (OO) 软件的体系结构是包含协作类的一系列分层的子系统。这些系统的每个元素 (子系统和类) 所执行的功能都有助于满足系统需求。有必要在各种不同的层次上测试面向对象系统, 尽力发现当类之间存在协作以及子系统穿越体系结构层通信时可能发生的错误。

人员: 面向对象测试由软件工程师和测试专家执行。

重要性: 在将程序交付给客户之前, 必须运行程序, 试图去除所有的错误, 使得客户免受糟糕软件产品的折磨。为了发现尽可能多的错误, 必须进行系统的测试, 并且必须使用严格的技术来设计测试用例。

步骤: 面向对象测试在策略上类似传统系统的测试, 但在战术上是不同的。面向对象分析和设计模型在结构和内容上类似于

最终的面向对象程序, 因此“测试”开始于对这些模型的评审。一旦代码已经生成, 面向对象测试就开始进行“小规模”的类测试。设计一系列测试以检查类操作及一个类与其他类协作时是否存在错误。当将类集成起来构成子系统时, 应用基于线程的测试、基于使用的测试、簇测试以及基于故障的测试方法彻底检查协作类。最后, 运用用例 (作为分析模型的一部分开发) 发现软件确认级的错误。

工作产品: 设计并文档化一组测试用例来检查类、协作和行为。定义期望的结果, 并记录实际结果。

质量保证措施: 测试时应改变观点, 努力去“破坏”软件! 规范化地设计测试用例, 并对测试用例进行周密的评审。

在第 18 章已经提到, 简单地说, 测试的目标就是在可行的时间期限内, 以可行的工作量发现最大可能数量的错误。虽然这个基本目标对于面向对象软件没有改变, 但面向对象程序的本质改变了测试策略和测试战术。

可以断定, 由于可复用类库规模的增大, 更多的复用会缓解对面向对象系统进行繁重测试的需求。然而确切地说, 相反的情况的确也是存在的。Binder[Bin94b] 在讨论这种情况时说道:

每次复用都是一种新的使用环境, 重新测试需要谨慎。为了在面向对象系统中获得高可靠性, 似乎需要更多的测试, 而不是更少的测试。

为了充分测试面向对象的系统, 必须做三件事情: (1) 对测试的定义进行扩展, 使其包括应用于面向对象分析和设计模型的错误发现技术; (2) 单元测试和集成测试策略必须彻底改变; (3) 测试用例设计必须考虑面向对象软件的独特性质。

关键概念

- 类测试
- 簇测试
- 一致性
- 基于故障的测试
- 多类测试
- 面向对象模型
- 划分测试
- 随机测试
- 基于场景的测试
- 测试方法
- 测试策略
- 基于线程的测试
- 基于使用的测试

19.1 扩展测试的视野

面向对象软件的构造开始于分析和设计模型的创建^①。由于面向对象软件工程模式的进化特性，这些模型开始于系统需求的不太正式的表示，并进化到更详细的类模型、类关系、系统设计和分配以及对象设计（通过消息传递来合并对象连接模型）。在每一个阶段，都要对模型进行“测试”，尽量在错误传播到下一轮迭代之前发现错误。

可以肯定，面向对象分析和设计模型的评审非常有用，因为相同的语义结构（例如，类、属性、操作、消息）出现在分析、设计和代码层次。因此，在分析期间所发现的类属性的定义问题会防止副作用的发生。如果问题直到设计或编码阶段（或者是分析的下一轮迭代）还没有发现，副作用就会发生。

例如，在分析的第一轮迭代中，考虑定义了很多属性的一个类。有一个无关的属性被扩展到类中（由于对问题域的错误理解），然后指定了两个操作来处理此属性。对分析模型进行了评审，领域专家指出了这个问题。在这个阶段去除无关的属性，可以在分析阶段避免下面的问题和不必要的工作量。

1. 可能会生成特殊的子类，以适应不必要的属性或例外。去除无关的属性后，与创建不必要的子类相关的工作就可以避免。
2. 类定义的错误解释可能导致不正确或多余的类关系。
3. 为了适应无关的属性，系统的行为或类可能被赋予不适当的特性。

如果问题没有在设计期间被发现以致于进一步传播，则在设计期间会发生以下问题（早期的评审可以避免这些问题的发生）。

1. 在系统设计期间，可能会发生将类错误地分配给子系统和任务的情况。
2. 可能会扩展不必要的设计工作，比如为涉及无关属性的操作创建过程设计。
3. 消息模型可能不正确（因为会为无关的操作设计消息）。

如果问题没有在设计期间检测出来，以致于传递到编码活动中，那么将大幅增加生成代码的工作量，用于实现不必要的属性、两个不必要的操作、驱动对象间通信的消息以及很多其他相关的问题。另外，类的测试会消耗更多不必要的时间。一旦最终发现了这个问题，一定要对系统执行修改，以处理由变更所引起的潜在副作用。

在开发的后期，面向对象分析（OOA）和面向对象设计（OOD）模型提供了有关系统结构和行为的实质性信息。因此，在代码生成之前，需要对这些模型进行严格的评审。

应该在模型的语法、语义和语用方面对所有的面向对象模型进行正确性、完整性和一致性测试（在这里，术语测试包括技术评审）。

建议 尽管面向对象分析和设计模型的评审是测试面向对象应用不可分割的一部分，但要认识到这是不够的，还要实施可运行的测试。

引述 我们使用的工具对我们的思考习惯具有深远的影响，因此，也对我们的思考能力具有深远的影响。

Edsger Dijkstra

19.2 测试 OOA 和 OOD 模型

不能在传统意义上对分析和设计模型进行测试，因为这些模型是不能运行的。然而，可以使用技术评审检查模型的正确性和一致性。

① 分析建模和设计建模技术在本书第二部分介绍。基本的面向对象概念在附录 2 中介绍。

19.2.1 OOA 和 OOD 模型的正确性

用于表示分析和设计模型的符号和语法是与为项目所选择的特定分析和设计方法连接在一起的。由于语法的正确性是基于符号表示的正确使用来判断的，因此必须对每个模型进行评审以确保维持了正确的建模习惯。

在分析和设计期间，可以根据模型是否符合真实世界的问题域来评估模型的语义正确性。如果模型准确地反映了现实世界（详细程度与模型被评审的开发阶段相适应），则在语义上是正确的。实际上，为了确定模型是否反映了现实世界的需求，应该将其介绍给问题领域的专家，由专家检查类定义以及层次中遗漏和不清楚的地方。要对类关系（实例连接）进行评估，确定这些关系是否准确地反映了现实世界的对象连接[⊖]。

19.2.2 面向对象模型的一致性

面向对象模型的一致性可以通过这样的方法来判断：“考虑模型中实体之间的关系。不一致的分析模型或设计模型在某一部分中的表示没有正确地反映到模型的其他部分” [McG94]。

为了评估一致性，应该检查每个类及其与其他类的连接。可以使用类－职责－协作者（Class-Responsibility-Collaborator，CRC）模型和对象－关系图来辅助此活动。如在第 9 章所学到的，CRC 模型由 CRC 索引卡片组成。每张 CRC 卡片都列出了类的名称、职责（操作）和协作者（接收其消息的其他类及完成其职责所依赖的其他类）。协作意味着面向对象系统的类之间的一系列关系（即连接）。对象关系模型提供了类之间连接的图形表示。这些信息都可以从分析模型（第 9 章）中获得。

推荐使用下面的步骤对类模型进行评估 [McG94]。

- 1. 检查 CRC 模型和对象－关系模型。对这两个模型做交叉检查，确保需求模型所蕴含的所有协作都已正确地反映在这两个模型中。
- 2. 检查每一张 CRC 索引卡片的描述以确定委托职责是协作者定义的一部分。例如，考虑为销售积分结账系统定义的类（称为 CreditSale），这个类的 CRC 索引卡片如图 19-1 所示。

类的名称：credit sale	
类的类型：transaction event	
类的特性：nontangible, atornio, esquential, pemanert, guarded	
职责：	协作者
读信用卡	信用卡
取得授权	信用权利
显示购物金额	产品票
	销售总账
	审计文件
生成账单	账单

图 19-1 用于评审的 CRC 索引卡片实例

⊖ 对于面向对象系统，在对照现实世界的使用场景追踪分析和设计模型方面，用例是非常有价值的。

对于这组类和协作,例如,将职责(例如读信用卡)委托给已命名的协作者(CreditCard),看看此协作者是否完成了这项职责。也就是说,类CreditCard是否具有读卡操作?在此实例中,回答是肯定的。遍历对象-关系模型,确保所有此类连接都是有效的。

3. 反转连接,确保每个提供服务的协作者都从合理的地方收到请求。例如,如果CreditCard类收到了来自CreditSale类的请求purchase amount,那么就有问题了。CreditCard不知道购物金额是多少。
4. 使用步骤3中反转后的连接,确定是否真正需要其他类,或者职责在类之间的组织是否合适。
5. 确定是否可以将广泛请求的多个职责组合为一个职责。例如,读信用卡和取得授权在每一种情形下都会发生,可以将这两个职责组合为验证信用请求(validate credit request)职责,此职责包括取得信用卡号和取得授权。

可以将步骤1~步骤5反复应用到每个类及需求模型的每一次评估中。

一旦创建了设计模型(第11~14章),就可以进行系统设计和对象设计的评审了。系统设计描述总体的产品体系结构、组成产品的子系统、将子系统分配给处理器的方式、将类分配给子系统的方式以及用户界面的设计。对象模型描述每个类的细节以及实现类之间的协作所必需的消息传送活动。

系统设计评审是这样进行的:检查面向对象分析期间所开发的对象-行为模型,并将所需要的系统行为映射到为完成此行为而设计的子系统上。在系统行为的范畴内也要对并发和任务分配进行评审。对系统的行为状态进行评估以确定并发行为。使用用例进行用户界面设计。

对照对象-关系网检查对象模型,确保所有的设计对象都包括必要的属性和操作,以实现为每个CRC索引卡片所定义的协作。另外,要对操作细节的详细规格说明(即实现操作的算法)进行评审。

19.3 面向对象测试策略

如在第17章讲到的,经典的软件测试策略从“小范围”开始,并逐步过渡到“软件整体”。用软件测试的行话来说(第18章),就是先从单元测试开始,然后过渡到集成测试,并以确认测试和系统测试结束。在传统的应用中,单元测试关注最小的可编译程序单元——子程序(例如,构件、模块、子程序、程序)。一旦完成了一个单元的单独测试,就将其集成到程序结构中,并进行一系列的回归测试,以发现模块的接口错误及由于加入新模块所引发的副作用。最后,将系统作为一个整体进行测试,确保发现需求方面的错误。

19.3.1 面向对象环境中的单元测试

考虑面向对象软件时,单元的概念发生了变化。封装是类和对象定义的驱动力,也就是说,每个类和类的每个实例(对象)包装了属性(数据)和操纵这些数据的操作(也称为方法或服务)。最小的可测试单元是封装了的类,而不是单独的模块。由于一个类可以包括很多不同的操作,并且一个特定的操作又可以很多不同类的一部分,因此,单元测试的含义发生了巨大的变化。

关键点 在面向对象软件中,最小的可测试“单元”是类,类测试是由封装在类中的操作和类的状态行为驱动的。

我们已经不可能再独立地测试单一的操作了（独立地测试单一的操作是单元测试的传统观点），而是要作为类的一部分进行操作。例如，考虑在一个类层次中，为超类定义了操作 $X()$ ，并且很多子类继承了此操作。每个子类都使用操作 $X()$ ，但是此操作是在为每个子类所定义的私有属性和操作的环境中应用的。由于使用操作 $X()$ 的环境具有微妙的差异，因此，有必要在每个子类的环境中测试操作 $X()$ 。这就意味着在真空中测试操作 $X()$ （传统的单元测试方法）在面向对象的环境中是无效的。

面向对象软件的类测试等同于传统软件的单元测试^①。面向对象软件的类测试与传统软件的单元测试是不同的，传统软件的单元测试倾向于关注模块的算法细节和流经模块接口的数据，而面向对象软件的类测试由封装在类中的操作和类的状态行为驱动。

19.3.2 面向对象环境中的集成测试

由于面向对象软件不具有层次控制结构，因此传统的自顶向下和自底向上的集成策略是没有意义的。另外，由于“组成类的构件之间的直接和非直接的交互”[Ber93]，因此每次将一个操作集成到类中通常是不可能的。

面向对象系统的集成测试有两种不同的策略 [Bin94a]。第一种集成策略是基于线程的测试，将响应系统的一个输入或一个事件所需要的一组类集成到一起。每个线程单独集成和测试，并应用回归测试确保不产生副作用。第二种集成策略是基于使用的测试，通过测试那些很少使用服务器类的类（称为独立类）开始系统的构建。测试完独立类之后，测试使用独立类的下一层类（称为依赖类）。按照这样的顺序逐层测试依赖类，直到整个系统构建完成。与传统集成不同，在可能的情况下，这种策略避免了作为替换操作的驱动模块和桩模块的使用（第 18 章）。

关键点 面向对象软件的集成测试是对响应一个给定事件所需要的一组类进行测试。

簇测试 [McG94] 是面向对象软件集成测试中的一个步骤。通过设计试图发现协作错误的测试用例，对一簇协作类（通过检查 CRC 和对象 - 关系模型来确定）进行测试。

19.3.3 面向对象环境中的确认测试

在确认级或系统级，类连接的细节消失了。如传统的确认方法一样，面向对象软件的确认关注用户可见的动作和用户可以辨别的来自系统的输出。为了辅助确认测试的导出，测试人员应该拟定出用例（第 8 章和第 9 章），用例是需求模型的一部分，提供了最有可能发现用户交互需求方面错误的场景。

传统的黑盒测试方法（第 18 章）可用于驱动确认测试。另外，测试人员可以选择从对象 - 行为模型导出测试用例，也可以从创建的事件流图（OOA 的一部分）导出测试用例。

19.4 面向对象测试方法

面向对象体系结构导致封装了协作类的一系列分层子系统的产生。每个系统成分（子系统和类）完成的功能都有助于满足系统需求。有必要在不同的层次上测试面向对象系统，以发现错误。在类相互协作以及子系统

引述 我将测试人员看成是项目的护卫。开发人员专注于创造成功，而测试人员则守护在他们左右，使其免遭失败。

James Bach

① 面向对象类的测试用例设计方法在 19.4 ~ 19.6 节讨论。

穿越体系结构层通信时可能出现这些错误。

面向对象软件的测试用例设计方法还在不断改进,然而,对于面向对象测试用例的设计,Berard已经提出了总体方法[Ber93]:

1. 每个测试用例都应该被唯一地标识,并明确地与被测试的类相关联。
2. 应该叙述测试的目的。
3. 应该为每一个测试开发测试步骤,并包括以下内容:将要测试的类的指定状态列表;作为测试结果要进行检查的消息和操作列表;对类进行测试时可能发生的异常列表;外部条件列表(即软件外部环境的变更,为了正确地进行测试,这种环境必须存在);有助于理解或实现测试的补充信息。

面向对象测试与传统的测试用例设计是不同的,传统的测试用例是通过软件的输入-处理-输出视图或单个模块的算法细节来设计的,而面向对象测试侧重于设计适当的操作序列以检查类的状态。

19.4.1 面向对象概念的测试用例设计含义

经过分析模型和设计模型的演变,类成为了测试用例设计的目标。由于操作和属性是封装的,因此从类的外面测试操作通常是徒劳的。尽管封装是面向对象的重要设计概念,但它可能成为测试的一个小障碍。如Binder[Bin94a]所述:“测试需要报告对象的具体状态和抽象状态。”然而,封装使获取这些信息有些困难,除非提供内置操作来报告类的属性值,否则,可能很难获得一个对象的状态快照。

网络资源 一些极好的有关面向对象测试的论文集和资源可在<https://www.thecsiac.com>找到。

继承也为测试用例设计提出了额外的挑战。我们已经注意到,即使已取得复用,每个新的使用环境也需要重新测试。另外,由于增加了所需测试环境的数量,因此多重继承^①使测试进一步复杂化[Bin94a]。若将从超类派生的子类实例用于相同的问题域,则测试子类时,使用超类中生成的测试用例集是可能的。然而,若子类用在一个完全不同的环境中,则超类的测试用例将具有很小的可应用性,因而必须设计新的测试用例集。

19.4.2 传统测试用例设计方法的可应用性

第18章描述的白盒测试方法可以应用于类中定义的操作。基本路径、循环测试或数据流技术有助于确保一个操作中的每条语句都测试到。然而,许多类操作的简洁结构使某些人认为:用于白盒测试的工作投入最好直接用于类层次的测试。

与利用传统的软件工程方法所开发的系统一样,黑盒测试方法也适用于面向对象系统。如我们在第18章提到的,用例可为黑盒测试和基于状态的测试设计提供有用的输入。

关键点 基于故障的测试策略是假设一组似乎可能出现的故障,然后导出测试去证明每个假设。

19.4.3 基于故障的测试^②

在面向对象系统中,基于故障的测试目标是设计测试以使其最有可能发现似乎可能出现的故障(以下称为似然故障)。由于产品或系统必须符合

① 应非常小心使用的一个面向对象概念。

② 19.4.3节和19.4.4节是从Brain Marick发布在因特网新闻组comp.testing上的文章中摘录的,已得到作者的许可。有关该主题的详细信息见[Mar94]。应该注意到,19.4.3节和19.4.4节讨论的技术也适用于传统软件。

客户需求，因此完成基于故障的测试所需的初步计划是从分析模型开始的。测试人员查找似然故障（即系统的实现中有可能产生错误的方面）。为了确定这些故障是否存在，需要设计测试用例以检查设计或代码。

当然，这些技术的有效性依赖于测试人员如何理解似然故障。若在面向对象系统中真正的故障被理解为“没有道理”的，则这种方法实际上并不比任何随机测试技术好。然而，若分析模型和设计模型可以洞察有可能出错的事物，则基于故障的测试可以花费相当少的工作量而发现大量的错误。

集成测试寻找的是操作调用或信息连接中的似然错误。在这种环境下，可以发现三种错误：非预期的结果，使用了错误的操作 / 消息，以及不正确的调用。为确定函数（操作）调用时的似然故障，必须检查操作的行为。

提问 在操作调用和消息连接中会遇到哪些类型的故障？

集成测试适用于属性，同样也适用于操作。对象的“行为”通过赋予属性值来定义。测试应该检查属性以确定不同类型的对象行为是否存在合适的值。

集成测试试图发现用户对象而不是服务对象中的错误，注意到这一点很重要。用传统的术语来说，集成测试的重点是确定调用代码而不是被调用代码中是否存在错误。以操作调用为线索，这是找出调用代码的测试需求的一种方式。

19.4.4 基于场景的测试设计

基于故障的测试忽略了两种主要类型的错误：（1）不正确的规格说明；（2）子系统间的交互。当出现了与不正确的规格说明相关的错误时，产品并不做客户希望的事情，而是有可能做错误的事情或漏掉重要的功能。但是，在这两种情况下，质量（对需求的符合性）均会受到损害。当一个子系统的行为创建的环境（例如事件、数据流）使另一个子系统失效时，则出现了与子系统交互相关的错误。

基于场景的测试关心用户做什么，而不是产品做什么。这意味着捕获用户必须完成的任务（通过用例），然后在测试时使用它们及其变体。

场景可以发现交互错误。为了达到这个目标，测试用例必须比基于故障的测试更复杂且更切合实际。基于场景的测试倾向于用单一测试检查多个子系统（用户并不限制自己一次只用一个子系统）。

关键点 基于场景的测试将发现任何角色与软件进行交互时出现的错误。

19.5 类级可应用的测试方法

“小范围”测试侧重于单个类及该类封装的方法。面向对象测试期间，随机测试和分割是用于检查类的测试方法。

19.5.1 面向对象类的随机测试

为简要说明这些方法，考虑一个银行应用，其中 Account 类有下列操作：open()、setup()、deposit()、withdraw()、balance()、summarize()、creditLimit() 及 close()[Kir94]。其中，每个操作均可应用于 Account 类，但问题的本质隐含了一些限制（例如，账号必须在其他操作可应用之前打开，在所有操作完成之后关闭）。即使有了这些限制，仍存在很多种操作排列。一个 Account 对象的最小行为的生命历史包含以下操作：

建议 随机测试可能的排列数可以变得相当大。与正交数组测试相类似的策略可以用于提高测试的有效性。

```
open•setup•deposit•withdraw•close
```

这表示 Account 的最小测试序列。然而，可以在这个序列中发生大量其他行为：

```
open•setup•deposit•[deposit|withdraw|balance|summarize|creditLimit]n•withdraw•close
```

可以随机产生一些不同的操作序列，例如：

测试用例 r_1 : open•setup•deposit•deposit•balance•summarize•withdraw•close

测试用例 r_2 : open•setup•deposit•withdraw•deposit•balance•creditLimit•withdraw•close

执行这些序列和其他随机顺序测试，以检查不同类实例的生命历史。

SafeHome 类测试

[场景] Shakira 的工作间。

[人物] Jamie 与 Shakira, SafeHome 软件工程团队成员，负责安全功能的测试用例设计。

[对话]

Shakira：我已经为 Detector 类（图 13-4）开发了一些测试，你知道，这个类允许访问安全功能的所有 Sensor 对象。你熟悉它吗？

Jamie（笑）：当然，它允许你加入“小狗焦虑症”传感器。

Shakira：而且是唯一的一个。不管怎么样，它包含 4 个操作的接口：read()、enable()、disable() 和 test()。在传感器可读之前，它必须被激活。一旦激活，就可以进行读和测试，且随时可以中止它，除非正在处理警报条件。因此，我定义了检查其行为生命历史的简单测试序列（向 Jamie 展示下述序列）。

```
#1: enable•test•read•disable
```

Jamie：不错。但你还得做更多的测试！

Shakira：我知道。这里有我提出的其他测

试序列。

（向 Shakira 展示下述序列）

```
#2: enable•test*[read]n•test•disable
```

```
#3: [read]n
```

```
#4: enable*disable•[test | read]
```

Jamie：看我能否理解这些测试序列的意图。#1 通过一个正常的生命历史，属于常规使用。#2 重复 read() 操作 n 次，那是一个可能出现的场景。#3 在传感器激活之前尽力读取它……那应该产生某种错误信息，对吗？#4 激活和中止传感器，然后尽力读取它，这与 #2 不是一样的吗？

Shakira：实际上不一样。在 #4 中，传感器已经被激活，#4 测试的实际上是 disable() 操作是否像其预期的一样有效工作。disable() 之后，read() 或 test() 应该产生错误信息。若没有，则说明 disable() 操作中有错误。

Jamie：太棒了，记住这 4 个测试必须应用于每一类传感器，因为所有这些操作根据其传感器类型可能略有不同。

Shakira：不用担心，那是计划之中的事。

19.5.2 类级的划分测试

与传统软件的等价划分（第 18 章）基本相似，划分测试（partition testing）可减小测试特定类所需的测试用例数量。对输入和输出进行分类，设计测试用例以检查每个分类。但划分类别是如何得到的呢？

基于状态的划分就是根据它们改变类状态的能力对类操作进行分类。例如，考虑 Account 类，状态操作包括 deposit() 和 withdraw()，而非状态操作包括

提问 在类级上什么测试选项可供使用？

balance()、summarize() 和 creditLimit()。将改变状态的操作和不改变状态的操作分开，分别进行测试，因此：

测试用例 p_1 : open•setup•deposit•deposit•withdraw•withdraw•close

测试用例 p_2 : open•setup•deposit•summarize•creditLimit•withdraw•close

测试用例 p_1 检查改变状态的操作，而测试用例 p_2 检查不改变状态的操作（除了那些最小测试序列中的操作）。

也可以应用其他类型的划分测试。基于属性的划分就是根据它们所使用的属性对类操作进行分类。基于类别的划分就是根据每个操作所完成的一般功能对类操作进行分类。

19.6 类间测试用例设计

当开始集成面向对象系统时，测试用例的设计变得更为复杂。在这个阶段必须开始类间协作的测试。为说明“类间测试用例生成”[Kir94]，我们扩展 19.5 节中讨论的银行例子，让它包括图 19-2 中的类与协作。图中箭头的方向指明消息传递的方向，标注则指明作为消息隐含的协作的结果而调用的操作。

引述 对于面向对象的开发，定义单元测试和集成测试范围的边界是不同的。在这个过程中，可以在很多点上设计和检查测试。因此，“设计一点儿，编码一点儿”变成了“设计一点儿，编码一点儿，测试一点儿”。

Robert Binder

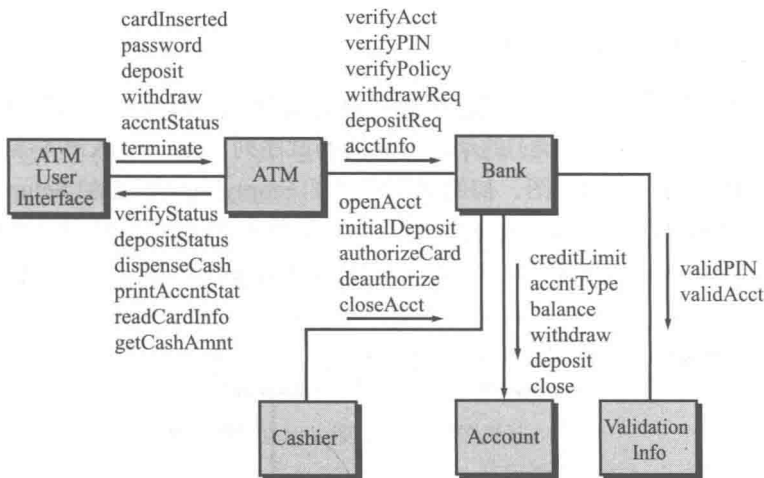


图 19-2 银行应用的类协作图 [Kir94]

与单个类的测试相类似，类协作测试可以通过运用随机和划分方法、基于场景测试及行为测试来完成。

19.6.1 多类测试

Kirani 和 Tsai[Kir94] 提出了利用下列步骤生成多类随机测试用例的方法：

1. 对每个客户类，使用类操作列表来生成一系列随机测试序列。这些操作将向其他服务类发送消息。
2. 对生成的每个消息，确定协作类和服务对象中的相应操作。
3. 对服务对象中的每个操作（已被来自客户对象的消息调用），确定它传送的消息。
4. 对每个消息，确定下一层被调用的操作，并将其引入到测试序列中。

为便于说明 [Kir94]，考虑 Bank 类相对于 ATM 类的操作序列（图 19-2）：

```
verifyAcct.verifyPIN.[[verifyPolicy.withdrawReq]|depositReq|acctInfoREQ]n
```

Bank 类的一个随机测试用例可以是：

```
测试用例 r3: verifyAcct.verifyPIN.depositReq
```

为考虑涉及该测试的协作者，考虑与测试用例 r₃ 中提到的操作相关的消息。为了执行 verifyAcct() 与 verifyPIN()，Bank 类必须与 ValidationInfo 类协作。为了执行 depositReq()，Bank 类必须与 Account 类协作。因此，检查这些协作的新测试用例为：

测试用例 r₄：

```
Test case r4 = verifyAcct [Bank:validAcctValidationInfo].verifyPIN  
[Bank: validPinValidationInfo].depositReq [Bank: depositaccount]
```

多个类的划分测试方法与单个类的划分测试方法类似，单个类的划分测试方法如在 19.5.2 节讨论的那样。然而，可以对测试序列进行扩展，以包括那些通过发送给协作类的消息而激活的操作。另一种划分测试方法基于特殊类的接口。参看图 19-2，Bank 类从 ATM 类和 Cashier 类接收消息，因此，可以通过将 Bank 类中的操作划分为服务于 ATM 类的操作和服务于 Cashier 类的操作对其进行测试。基于状态的划分（19.5.2 节）可用于进一步细化上述划分。

19.6.2 从行为模型导出的测试

在第 10 章中，我们已讨论过用状态图表示类的动态行为模型。类的状态图可用于辅助生成检查类（以及与该类的协作类）的动态行为的测试序列。图 19-3[Kir94] 给出了前面讨论的 Account 类的状态图。根据该图，初始变换经过了 Empty acct 状态和 Setup acct 状态，该类实例的绝大多数行为发生在 Working acct 状态。最终的 Withdrawal 和结束账户操作使得 Account 类分别向 Nonworking acct 状态和 Dead acct 状态发生转换。

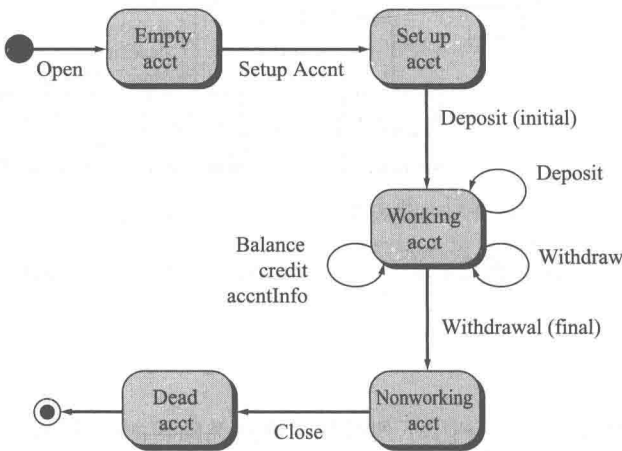


图 19-3 Account 类的状态转换图 [Kir94]

将要设计的测试应该覆盖所有的状态，也就是说，操作序列应该使 Account 类能够向所有可允许的状态转换：

```
测试用例 s1: open.setupAcct.deposit (initial).withdraw (final).close
```

应该注意到, 这个序列与 19.5.2 节所讨论的最小测试序列相同。下面将其他测试序列加入最小测试序列中:

测试用例 s_2 : `open•setupAcct•deposit(initial)•deposit•balance•credit•withdraw
(final)•close`

测试用例 s_3 : `open•setupAcct•deposit(initial)•deposit•withdraw•acctInfo•withdr
aw (final)•close`

可以设计更多的测试用例以保证该类的所有行为已被充分检查。在该类的行为与一个或多个类产生协作的情况下, 可以用多个状态图来追踪系统的行为流。

可以通过“广度优先” [McG94] 的方式来遍历状态模型。在这里, 广度优先意味着一个测试用例检查单个转换, 之后在测试新的转换时, 仅使用前面已经测试过的转换。

考虑银行系统中的一个 CreditCard 对象。CreditCard 对象的初始状态为 undefined (即未提供信用卡号)。在销售过程中一旦读取信用卡, 对象就进入了 defined 状态, 即属性 card number、expiration date 以及银行专用的标识符被定义。当信用卡被发送以请求授权时, 它处于 submitted 状态, 当接收到授权时, 它处于 approved 状态。可以通过设计使转换发生的测试用例来测试 CreditCard 对象从一个状态到另一个状态的转换。对这种测试类型的广度优先方法在检查 undefined 和 defined 之前不会检查 submitted 状态。若这样做了, 它就使用了尚未经过测试的转换, 从而违反了广度优先准则。

习题与思考题

- 19.1 用自己的话描述为什么在面向对象系统中类是最小的合理测试单元。
- 19.2 若现有类已进行了彻底的测试, 为什么我们还是必须对从现有类实例化的子类进行重新测试? 我们可以使用为现有类设计的测试用例吗?
- 19.3 为什么“测试”应该从面向对象分析和设计开始?
- 19.4 为 SafeHome 导出一组 CRC 索引卡片, 按照 19.2.2 节讲述的步骤确定是否存在不一致性。
- 19.5 基于线程和基于使用的集成测试策略有什么不同? 簇测试如何适应?
- 19.6 将随机测试和划分方法运用到设计 SafeHome 系统时定义的三个类。产生用于展示操作调用序列的测试用例。
- 19.7 运用多类测试及从 SafeHome 设计的行为模型中生成的测试。
- 19.8 运用随机测试、划分方法、多类测试及 19.5 节和 19.6 节所描述的银行应用的行为模型导出的测试, 再另外生成 4 个测试。

扩展阅读与信息资源

第 17 章和第 18 章的推荐读物与阅读信息部分所列出的很多测试方面的书都在一定程度上讨论了面向对象系统的测试。Bashir 和 Goel (《Testing Object-Oriented Software》, Springer, 2012)、Schach (《Object-Oriented and Classical Software Engineering》, McGraw-Hill, 8th ed., 2010) 以及 Bruege 和 Dutoit (《Object-Oriented Software Engineering Using UML, Patterns, and Java》, Prentice Hall, 3rd ed., 2009) 在更广的软件工程实践环境中考虑面向对象测试。Jorgensen (《Software Testing: A Craftsman's Approach》, Auerbach, 3rd ed., 2008) 讨论了形式化技术和面向对象技术。Yurga (《Testing and Testability of Object-Oriented Software Systems via Metrics: A Metrics-Based Approach to the Testing Process and Testability of Object-Oriented Software Systems》, LAP Lambert, 2011)、Sykes 和 McGregor (《Practical Guide to Testing

Object-Oriented Software》, Addison-Wesley, 2001)、Binder (《Testing Object-Oriented Systems》, Addison-Wesley, 1999) 以及 Kung 和他的同事 (《Testing Object-Oriented Software》, Wiley-IEEE Computer Society Press, 1998) 都非常详细地描述了面向对象测试。Freeman 和 Pryce (《Growing Object-Oriented Software, Guided by Tests》, Addison-Wesley, 2009) 讨论了面向对象软件的测试驱动设计。Denney (《Use Case Levels of Test: A Four-Step Strategy for Building Time and Innovation in Software Test Design》, CreateSpace Independent Publishing, 2012) 讨论了可以应用于测试面向对象系统的技术。

从网上可以获得大量有关面向对象测试方法的信息。最新的有关测试技术的参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

安全性工程

要点浏览

概念：安全性工程师构建的是具有保护他们的资产免受攻击能力的系统。使用威胁分析可以确定在系统的弱点受到攻击时所需的控制措施，以减少显露度。安全性是软件质量要素的重要前提条件，比如完整性、可用性、可靠性和安全性都以安全为前提。

人员：软件工程师要和依赖系统成果或依赖于服务的客户以及其他利益相关者合作。

重要性：每一门新兴技术都可能引起用户对于隐私的关注，并且为其有价值的信息可能被窃取而担忧。安全性不只是软件开发人员的关注点，更是军事部门、政府机构以及卫生部门的关注焦点。如今每一位软件工程师都必须关注安全性，他们一定要注意保护好项目客户的资源。

步骤：首先识别系统资产，确定由于安全性漏洞造成的损失。在构件层上建立系统结构模型。然后，制定安全性需求规格说明和风险缓解计划。在系统建成后，实施安全性保证，并将其贯彻于软件过程的始终。

工作产品：主要的工作产品是安全性规格说明（可能是需求模型的一部分）和作为系统质量保证文档一部分的文档化安全性用例。为了开发这些工作产品还要建立威胁模型，并制定安全性评估计划以及风险缓解计划。

质量保证措施：使用安全性评审和安全检查所得结果以及测试结果作为安全性用例，使系统的利益相关者可评估系统在保护资产和维护私密方面的可信任度。

Devanbu 和 Stubblebine[Dev00] 对他们提出的安全性工程路线图曾做出如下说明：

是否有一种软件系统不再需要安全了呢？事实是：从互联网可知晓的在个人计算机上运行的客户端应用程序，到可通过互联网访问的复杂的电信系统和电力系统，再到具有版权保护机制的商用软件，几乎所有受软件控制的系统均会面临潜在对手的威胁。软件工程师必须意识到这种威胁，并且要设计出具有可靠防卫性的系统，同时还要为客户提交有价值的产品。

十年前就有人撰文讨论威胁问题，而后伴随着网络的爆炸式增长、无处不在的应用系统和云的广泛使用，这类论文更是蜂拥而至。所有这些技术都提出了关于用户隐私和个人信息可能丢失或是被窃取的新的忧虑。安全性问题不仅是软件开发人员的关注点，更是国防部门、政府和卫生机构关注的焦点。如今，每位软件工程师都必须关注安全性问题，以切实保护客户端的资源。

关键概念

- 资产
- 保证用例
- 攻击
- 模式
- 面
- 显露度
- 安全性
- 保证
- 用例
- 工程
- 模型
- 需求
- 威胁
- 分析
- 建模
- 信任验证

从最简单的道理来说,软件的安全性提供了使软件系统保护资产免于受到攻击的机制。对于任一利益相关者而言,资产是具有价值的资源。资产包括数据库信息、文件、程序、硬盘驱动器的存储空间、系统内存甚至处理器的容量。攻击通常是利用软件的弱点或漏洞,致使未授权人访问系统。例如,在允许用户访问有价值的系统资源之前没有发现的问题很可能会成为一个漏洞。

软件安全性是软件质量保证(第16章)的一个方面。从本书前面的章节中我们已经知道,质量是不能因响应隐错报告才加入系统中的。与此相似,安全也很难因响应已发现的系统漏洞而加入现有的系统中[Gho01]。安全性问题必须要在软件过程的开始就予以考虑,将其纳入软件设计之中,作为编码工作的一部分来实现,并且在测试和部署过程中加以验证。

本章中,我们提供了一份关于软件安全性工程中重要问题的调查。当然,对于这一主题的全面讨论已经超出了本书的范围。更多的信息可在[All08]、[Lip10]和[Sin08]中查阅。

20.1 安全性需求分析

软件的安全性需求是由以下两方面确定的:一是与客户合作共同识别出的必须得到保护的资产;二是当出现安全性漏洞时,这些资产受损的成本。资产损失的价值被称为显露度。损失可用恢复或重建资产的时间和成本来度量。无关紧要的资产无需保护。

构建安全系统的重要组成部分是可能被用于破坏系统资源的预计条件,或是可能被用于破坏系统资源的威胁,或是使授权用户无法访问的威胁。这一过程称为威胁分析。在系统资产、系统漏洞和威胁识别出以后,便可以制定控制措施,使系统既可避免受到攻击,又可缓解所遭受的破坏。

软件安全性是软件的完整性、可用性、可靠性和安全性(第15章)的必要前提,要想创建能防御资产免受所有可能威胁的系统是做不到的。因此必须鼓励用户维护好关键数据和冗余系统构件的备份副本,确保其安全保密。

引述 除非认识不足,否则安全性问题始终受到人们的极度重视。

Robbie Sin-clair

建议 一定要关注具有最高价值和最大风险显露度的资产。

SafeHome 利益相关者的安全性

[场景] 软件工程团队的工作区。

[人物] Jami Lazar、Vinod Raman、Ed Robbins, 软件团队成员; Doug Miller, 软件工程经理; Lisa Perez, 营销团队成员兼产品工程代表。

[对话]

Vinod: 如果没有异议,由我来主持本次会议,可以吗?(在场各位都点头表示同意。)我们需要开始确定 SafeHome 项目的安全性问题。

Doug: 我们是否可以首先把大家所担心的那些防护事项列举出来?

Jamie: 好,如果一个外部黑客侵入 SafeHome 系统,企图抢劫或破坏主人的房子,会是怎样?

Lisa: 如果有人知道我们的系统不能抵御黑客入侵,公司的声誉就将受损。

Jamie: 暂且不说责任,那样的事件发生了也会被认定是设计有缺陷。

Doug: 在传输密码时,产品的网站界面有可能使外人截获密码。

Ed: 更重要的是,网站界面需要包含客户信息的数据库,所以我们有对保密问题的担忧。

Vinod：也许这是个好时机，让每个人花十分钟的时间，列出各自认为受到攻击时可能会丢失或损坏的资产。

(10分钟以后)

Vinod：好了，让我们把这些资产都贴在白板上，看看是否有类似的担忧。

(15分钟后，白板上已经贴满了)

Lisa：看起来存在很多问题，那么我们怎

么处理好呢？

Doug：需要根据资产损失所造成破坏的修复成本，把列出的各项进行优先排序。

Lisa：怎么做呢？

Vinod：我们需要用到历史项目数据，获取替换损失资产的实际成本。**Lisa**你要和法律部门联系，以得到我们可能要承担的赔偿责任。

20.2 网络世界中的安全性与保密性

互联网活动把传统的桌面浏览转移到场景中，其中浏览器提供了定制的动态内容。在合并了第三方网站数据的社区论坛上，用户可以输入自己的内容。很多桌面应用程序都利用网站浏览器界面访问本地数据，并在多种计算平台上提供相同的用户体验。因此，要求网站开发者提供更好的控制 and 安全性机制。不应为不可信源的数据和代码给予与可信程序员的代码相同的权限。为了使网站浏览器成为有效的用户界面，应该把保密、信任和安全当作最为重要的质量属性 [Sei11]。

在很多情况下，用户的机密信息流在因特网上都会跨越组织的边界。例如，患者的电子信息可能需要在医院、保险公司和医生之间共享。同样，旅游信息可能需要在旅行社、酒店和航空公司之间共享。在这些情况下，用户必须披露自己的个人信息以接受服务。当这些信息以数字形式被接收以后，用户通常无法控制组织用它做什么。在理想情况下，用户应当可以直接控制他们的数据信息以进行处理和共享，但是当这些数据电子化后，就需要由用户规定数据共享的优先权 [Pea11]。

引述 依靠政府来保护你的隐私，就像要求一个偷窥狂来帮你安装百叶窗一样。

John Perry
Barlow

20.2.1 社交媒体

在线社交媒体网络的迅速增长和普及，使它们成为吸引恶意程序人员的目标。由于默认被信任的大多数用户都处于社交网络环境中，因此黑客很容易利用受损的账户向账户持有人的朋友发送恶意的已感染信息。社交网络可能被用以引诱用户到钓鱼网站[⊖]，欺骗他们提交个人资料或是转发现金给急需的朋友。另一诡计则是利用含有详细信息的电子邮件窃取用户的个人信息 [Sae11]。

引述 只要你把隐私披露出来，就不应该责备别人将其泄露出去。

Kahlil Gibran

有些社交媒体网络允许用户开发自己的应用程序。这些应用程序只允许用户转让个人信息，而后以用户意想不到的方式使用这些信息。有些应用程序或是游戏足以吸引知识渊博的用户提供他们的个人信息，以便使用该应用程序。社交网络往往有签到功能，这就使得罪犯能够瞄准用户在现实生活中的活动作案。无论是身份盗取、垃圾邮件还是间谍软件问题，许多计算机用户都未能采取积极措施进行自我保护 [Sta10]。

⊖ 钓鱼网站伪装成知名且值得信任的网站，引诱用户提供个人信息，可导致安全资产的损失。

20.2.2 移动 App

移动 App 的用户可以使用与固定有线网络用户几乎相同的网络服务。无线互联网用户除继承了所有与桌面商务相关的安全风险以外，还外加了移动网络特有的新风险。无线网络要求节点之间的信任与合作。这可能会被恶意程序所利用而拒绝服务或收集机密信息。利用设备拥有者的所有权限，为移动设备开发的平台和语言都可能被黑客攻击，并且恶意代码也可能被插入设备的系统软件之中。这就意味着安全性技术（例如登录、身份验证和加密）都可能很容易地受到破坏。

提问 我们在移动 App 中会遇到什么威胁？

20.2.3 云计算

由于数据是委托给服务提供商管理的远程服务器的，因此云计算自然地带有更多的机密性和隐私问题。这些云服务的提供商拥有全面访问和控制我们信息的能力。相信他们不会与别人分享这些信息（无论是有意或是无意的），并且会采取负责的措施来防止发生损失。对于数据挖掘而言，在线数据存储库是非常诱人的信息源（例如，收集人口统计信息或市场信息等）。问题是数据始发者并没给出同意的信息 [Ray11a]。因此，策略制定者应该制定出政策和法规，以确保服务提供商不滥用用户的信任。

当一家公司采用云计算时，“内部置信”与“外部不置信”之间的边界会变得模糊。由于组织的应用程序和数据不再处于原位，因而可能出现一种新型的内部恶意人员。机密数据可能只用几条命令就被恶意者或是无资格的系统管理员非法访问。大多数云服务提供商在适当环境下均有严格的规定来监督员工访问客户数据。但针对远程攻击与监控时，防止数据遭受不法物理访问的策略并不高效，往往只是在事发之后才能检测到攻击。在云系统中，为取得用户的信任，为用户提供一些评估保护机密性和隐私的必要机制是否合适的手段是很重要的 [Roc11]。

无处不在的网络访问和云计算的出现为商业合作提供了全新的形式。但共享信息和机密的保护却是一项艰巨的任务。安全的多方计算提高了利己行为的风险，除非各方都有信心保证没有人会利用系统谋取私利。这种情况突出了人们对系统信任和安全性的心理空间的维度，不能只靠软件工程来解决 [Ker 11]。

20.2.4 物联网

富有想象力的人士是这样来描述“物联网”的 [Rom11]：物联网上的任何实在的东西在互联网上都有其虚拟的实体。这些虚拟实体可以提供服务，也可以消费服务，并且朝着一个共同的目标进行合作。例如，通过用户所用周边设备的网络，从一部手机上就可以了解到该用户的身体状况和精神状况；汽车工程师设想的轿车可以和其他车辆、数据源和设备之间进行自主通信，而无需驾驶员的直接控制。

然而，安全性是横亘在通往这个愿景道路上的一个主要障碍。如果没有强大的安全性基础，攻击和故障将会超过物联网的任何优势。策略的制定者必须考虑治理与创新之间的平衡。过度的治理可能很容易阻碍创新，反之，创新又可能在不经意间忽视人权 [Rom11]。

20.3 安全性工程分析

安全性分析包括需求获取、威胁建模、风险分析、测度设计和正确性检查。除去商业理由以外，这些任务包括系统的功能性和非功能性细节的考量 [Bre03]。

20.3.1 安全性需求获取

本书第7章讨论的需求获取的通用技术同样适用于安全性需求的获取。安全性需求是非功能性需求^①，它常常影响着软件系统的体系结构设计。使用威胁建模和风险分析将系统需求进行精炼和优化以后，就可以制定系统的安全策略了。除去考虑使用情况以外，为获取所需的安全性体系结构，将采用安全性建模和分解的方法，使这些策略得到细化。在这些策略实施前体系结构的安全性方面已得到确认 [Bod09]。

有些情况下安全性需求和其他需求是相互矛盾的。例如，安全性和可用性之间就有可能相互冲突。高度安全的系统会使那些没有太多经验的用户感到难以使用。在以用户为中心的安全性工程中，安全性需求获取对三个重要问题给出了回答 [Mar02]。这三个问题是：（1）对于安全性软件，用户的要求是什么？（2）如何设计安全体系结构，使其可提供良好的用户界面设计？（3）如何设计良好的用户界面，使软件不仅安全性好，同时还能使它运行起来有效、高效，并且让用户满意？应该把这些问题的答案与（第7章中讨论的）利益相关者和系统资源间的交互作用结合起来考虑。

在实施需求获取时，分析师应首先认清攻击模式。攻击模式是用于识别系统安全性缺陷的一种设计模式。通过为常见的安全性漏洞提出问题和解决方案，攻击模式可加速安全性分析。复用攻击模式能帮助工程师识别系统漏洞。但没有必要重新设计不同的方式来攻击系统。针对软件安全性问题，攻击模式允许开发者使用易于理解的名称（例如，网络钓鱼、SQL注入和跨网站脚本等）。常用的攻击模式可随时间的推移而得到改善 [Sin08]。当应用特定的模式时，使用攻击模式的困难之处是尽人皆知的。

一些软件工程师认为，在敏捷过程中安全性工程的严格性与需求获取的非正式特征是不相容的（第5章）。然而，有一项可用于调解“正式差距”的技术是在需求域内给出滥用者故事。滥用者故事基于客户输入来描述对系统资产的威胁。滥用者故事扩展了已建立的用户故事这一有效的敏捷概念，并且有助于实现安全性需求的可追踪性，使安全性保证得以持续进行 [Pee11]。

引述 用户在安全性问题上往往表现得很迟钝。

Bruce Schneier

提问 我们应该提出哪些关于安全性需求获取的问题？

引述 防卫计算机系统历来是一场智力的战斗，入侵者企图找到漏洞，而设计师们则设法封堵漏洞。

Gosser

20.3.2 安全性建模

建模是说明需求和分析需求的一个重要过程。安全性模型是软件系统安全性策略的形式化描述。安全性策略提出了系统安全性的定义。安全性策略捕捉关键的系统安全性需求，同时将描述系统运行过程中如何加强安全性的规则包含在其中。

在设计、编码和评审过程中，安全性模型可以提供精确的指导。在系统建成之后，安全性模型提供了帮助验证安全性实施正确性的基础 [Dan09]。在维护活动中，安全性模型也是系统演化升级或维修的有价值的参考。

安全性模型可以用文字或图形来表示。无论安全性模型的表述形式如何，它都需要包含以下内容：（1）安全性策略的目标；（2）外部界面的需求；（3）软件安全性需求；（4）运行规则；（5）描述模型与系统对应关系的详细说明。

提问 安全性模型中包括哪些信息？

① 有时称其为横切关注点，在第4章中曾进行了讨论。

有些安全性模型用状态机表示^①。每一状态都必须包括与系统安全性相关的信息，作为一名与安全性相关的软件工程师必须确保系统的任何状态都在安全状态下启动，并且在安全状态下结束。还必须验证初始系统状态是安全的。为了帮助人们完整地理解，模型需要有说明，以表明模型和实际系统的关系。

在验收之前，可执行的建模形式可让开发者验证安全性模型及其行为。在验收之后，模型便成为设计的良好基础。用于安全性需求建模的两种语言是 UML.sec（这是用构造型和约束对 UML 所做的扩展）和 GRL（用于捕捉非功能性需求的面向目标的需求语言）。在开发系统时形式化建模语言的使用有助于提高系统的可信度 [Sal11]。

作为系统的安全性分析和验证的扩展手段，人们提出了形式化方法。在基于模型的安全性测试中，使用安全性需求的形式化规格说明可以帮助生成测试用例。对于关键的系统构件使用形式化证明可以增强开发者的信心，因为系统确实符合其规格说明。当然必须十分谨慎，要使证明的基本假设都得到满足。

20.3.3 测度设计

为了实现安全性，软件必须具有三种属性：可靠性（软件可以在不友好的环境下运行）、可信性（系统不会在恶意的方式下运行）和存活性（在已妥协的情况下系统可继续运行）^②。安全性度量^③和测度需要重点评估这些属性。

关键点 安全软件必须显示出三个属性：可靠性、可信性和存活性。

有用的安全性度量必须以测度为基础，使开发人员能够评估处于风险中的数据机密性和系统完整性可能达到的程度。生成此度量所需的三项测量是资产价值测度、威胁似然性测度和系统漏洞测度。这些属性都不易直接测量。损失资产的成本可能超过重建的成本。

最佳测度是在软件开发或运行期间现成的可用测度。办公桌上的安全投诉数量或安全性测试用例的失效数量可以提供一些测度（例如，每月报告的身份被盗事故的数量）。在攻击事件发生之前，我们可能并不知道有漏洞，但攻击得逞的数量是可以知道的。

20.3.4 正确性检查

安全性的正确性检查需要贯穿于整个软件开发周期。从对系统漏洞攻击的角度来分析，利益相关者资产的显露度应在开发过程的早期确定下来。

接着，软件团队要确保由系统用例导出的威胁模型已对风险缓解、风险监测和风险管理计划的安全性部分做出了解释。为在建模和构建活动期间使用，质量保证活动应包括安全性标准的制定和安全性指南的开发，软件验证活动应确保安全性测试用例是完备的，并可追溯到系统的安全性需求。

许多这类安全性检查均应包括在内植于常规软件工程任务的审核、审查和测试活动中（20.6 节）。正如在 20.4 节所述，对于在这些检查期间收集到的数据进行分析，并且概括为系统安全性用例的一部分。可信性验证过程见 20.7 节的讨论。

引述 理论上，我们可以建立可证明的安全系统。并且理论上也可以认为理论能够应用于实践。但实际上，这是不可能的。

M. Dacie

① 有限状态机是由一系列每一当前状态的可能转换状态和每个转换的触发条件定义的。

② <https://buildsecurityin.us-cert.gov>。

③ 度量是系统构件或过程具有特定属性的量化指标。良好的度量应满足 SMART（即特定的、可测量的、可得到的、可复用的和依赖时间的）标准。度量通常是利用统计技术来展示关系，通过实施测量而得到。

20.4 安全性保证

如今软件已融入了我们的日常生活，安全上的缺陷和与之相关的损失变得更为昂贵，同时也更为危险。完善的软件工程实践包含明确的需求和相应设计的开发，从而表明确已开发出适用的产品。安全性保证是为了向最终用户和其他利益相关者表明确已开发出一个安全产品，从而增强他们的信心。

20.4.1 安全性保证过程

验证是保证任务的一部分，它提供证据以表明利益相关者可以确信这款软件是符合需求的。在安全性工程的环境下考虑问题时，选择安全性需求的关键子集或软件的索赔要求，并提出保证案例，以证明该软件是能满足这些需求和索赔要求的。

保证案例是已经讨论过和审查过的材料，它所支持的论点是，软件可满足正在维权的索赔。保证案例曾长期用于软件的安全性(safety)，目前则正在用于软件的安全性(security)^①。于是常常称作安全性用例。

每个安全性用例包含三个要素：(1) 索赔要求本身；(2) 通过证据和假设使多个索赔要求之间彼此相连的论点；(3) 支持论点的证据主体和明确的假设。

为使安全性用例是有效的，三个目标必须得到满足：必须说明索赔要求对于该系统来说是适当的和可负担的；适用于工程实践的文件已得到实践，因而索赔要求是可以完成的；表明索赔的成果在风险要求的等级之内 [Red10]。

几种证据类型可用来证明安全性用例。如果用证明其正确性的意图来设计代码，那么软件正确性的形式化证明或许是有帮助的。有些工具支持自动软件验证 [DSi08]，而另一些工具则实施软件安全性漏洞的静态扫描(例如，RATS、ITS4、SLAM)^②。但是工具本身不能构建安全性用例。

一些证据可从对系统制品做类似于正式技术评审或审查得到。然而，这些评审只关注安全性的索赔要求。有安全性知识专长的人员可以评审系统或是安全性用例。检查单评估也可用来验证安全性指南和过程是否得到实施。

关键点 安全性用例支持索赔要求，因此说明这样的软件是安全可靠的。

SafeHome 构建安全性用例

【场景】 软件工程团队的工作区。

【人物】 Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员；Bridget Thornton，软件质量组组长。

【对话】

Ed： Bridget，感谢您来参加我们的讨论，我们正要为 SafeHome 项目构建安全性

用例。

Vinod： 我们从哪里开始讨论呢？

Bridget： 我们可以挑选一个存在安全性隐患的部件，看看我们能够找到什么证据来支持这个说法。

Ed： 是什么证据呢？

Bridget： 还是先挑选存在安全性隐患的

① 本章所涉及的 security 与 safety 的差别在于，前者包含保守秘密的意思，因此有人称其为“安密性”或“保密性”。——译者注

② 安全性工具的清单请参见 <http://www.tech-fag.com/how-to-find-security-vulnerabilities-in-source-code.html>。

部件吧!

Vinod: 让我们集中在与客户数据库有关的安全性薄弱点上。

Bridget: 好, 让我们从所列出的访问数据库的索赔要求开始吧!

Jamie: 你的意思是安全性模型的有些成分与数据库有关联?

Bridget: 是的, 接下来我们看看已完成的审查工作, 还有就是正式技术评审的摘要, 这个摘要应该是在完成这个项目里程碑时就做好的。

Ed: 那么过程审核和小组提交的变更请求文件又怎么考虑呢?

Bridget: 这些都很重要, 最好也考虑在内。

Vinod: 由独立测试组 (ITG) 生成并且运行多数系统测试用例。

Bridget: 归纳安全性测试用例的特点, 比较其预期的输出和实际的输出, 就可以得到安全性用例的非常重要的组成部分。

Jamie: 这里面可能有大量的信息需要处理。

Bridget: 是的, 这正是下一节我们为什么为了数据库安全性提出各项索赔要求, 并且总结出证据来支持或反驳有足够资产保护的索赔要求。

Ed: 在汇总材料时, 您能够帮助我们评审安全性用例吗?

Bridget: 当然可以。这个项目启动以后无论是进度超前还是滞后, 我们小组都要和你们团队进行持续沟通。

20.4.2 组织和管理

由于急着要把软件推向市场, 使得项目经理往往更为关注项目的特性和功能, 而把安全性列入次要地位。软件工程师应当重视软件设计和体系结构的健壮性。但除此之外, 在构建基于软件的系统时, 应当采用更为安全的做法 [Sob10]。

安全性保证和安全性风险识别活动要做计划、管理和跟踪, 这和其他软件工程活动是一样的。软件团队要收集数据 (例如, 非法访问数、系统失效数、数据记录丢失数等), 以确定什么活动起作用, 什么活动不起作用。这就要求开发人员分析所报告的每一个失效 (以确定失效的原因是否与系统漏洞相关), 然后评估所导致的资产显露度。

20.5 安全性风险分析[⊖]

识别和管理安全性风险是重要的项目计划任务 (第 22 章)。安全性工程是由软件团队和其他利益相关者识别出的风险所驱动的。风险影响着项目管理和安全性保证活动。

威胁建模是一种安全性分析方法, 可用于识别那些最有可能引发基于软件系统的破坏的威胁。威胁建模是在项目的初期阶段利用需求和分析模型完成的。建立威胁模型的工作包括: 识别应用的关键构件、分解应用、构件威胁的识别和分类、对每个构件威胁的分级与分类、根据风险大小的排序将构件进行分级以及制定缓解风险的策略。微软采用以下步骤生成威胁模型 [Sin08]:

构建威胁模型需要什么步骤?

1. **确认资产**。列出所有的敏感信息和知识产权、存储位置、存储方式以及谁有访问权。

关键点 威胁建模是一种安全性的分析方法, 可用于识别那些最有可能引发破坏的威胁。

⊖ 对软件项目风险分析的一般性讨论包括危及项目的所有类型的风险。其成功的分析方法在第 26 章中给出。

2. 给出体系结构概述。写出系统用例并建立系统构件模型。
3. 分解应用。目标是保证在应用构件之间发送的所有数据都是有效的。
4. 确认威胁。使用如攻击树或攻击模式等方法,记录可能危及系统资产的所有威胁,其过程往往包括寻找网络、主系统配置和应用威胁等。
5. 记录威胁。制作一个风险信息表,详细列出要监测和缓解的每项威胁。
6. 评估威胁。对于处理可能的威胁,多数项目所提供的资源都显得不足,因此要根据其影响大小和发生的可能性做出排序,以便区别对待。

贵重的资产应该得到更好的保护,以防高概率风险的破坏。量化风险评估过程(第26章)可用于风险排序。首先需确认所有要评估的资产,并且要确定损失或恢复重建的资金价值。对每一项资产列出主要的威胁表,并用历史数据来确定在一个典型年份里可能发生的每项威胁。针对每项资产计算出每年每项主要威胁可能损失的金额以及每年损失的预期值(Annual Loss Expectancy, ALE)。最后,将与每个单独的威胁相关的ALE值相加,计算出损失每项资产的综合威胁。

SafeHome 安全性工作步骤

[场景] 软件质量保证组工作区。

[人物] Jamie Lazar、Vinod Raman, 软件团队成员; Bridget Thornton, 软件质量组组长。

[对话]

Vinod: 嗨! Bridget 希望我们讨论安全性风险分析。

Bridget: 讨论这个有助于建立开发工作的安全性优先级吗?

Jamie: 我认为是的。

Vinod: 我们是否可以着眼于数据库的安全问题?

Jamie: 是的,通过历史数据,我们得知成本是备份数据和维护数据的记录。但是大家却不知道,如果客户的数据被盗,我们可能不知道应该给予的责任损害赔偿是多少,尽管对于这些成本我们有行业数据。

Jamie: 那就是我们要求的吗?

Bridget: 是的,你已经有了系统体系结构图。要验证在已被确认的构件之间交换的所有数据是比较容易的。我们还必须确定

每项资产的威胁。

Vinod: 那么该怎么做呢?

Bridget: 我们可以创建一个攻击树,从在根部设置攻击目标开始。例如,攻击者的目标可以是窃取客户信息。

Vinod: 还有呢?

Bridget: 然后查看数据库的攻击模式目录,找出合适的可作为攻击树的子目标。

Jamie: 然后怎么做?

Bridget: 你得要细化威胁,然后为每个威胁列出风险信息表,描述威胁的影响,任何监控措施和缓解措施都应到位,这样才能使问题得到解决。

Vinod: 那么如何做才能有助于建立开发的优先性顺序呢?

Bridget: 利用历史数据,对每一项威胁计算每年损失的预期值(ALE),便可确定每项威胁的成本,这部分过程我们可以提供帮助。

Jamie: 多谢, Bridget。我们在识别和精选威胁后,作ALE计算时会寻求您的输入信息。

20.6 传统软件工程活动的作用

在发现系统运行低效且经常出现故障之后,就需要考虑构建新系统,使其安全运行。然而,有些软件开发人员认为,直到威胁明显暴露以前,系统中的威胁是无法预测的。因此,直到测试阶段,他们总是忽视安全性问题。这样做只是靠一时填补漏洞,以清除在软件过程早期阶段形成的安全性失误。以特定的方式对已有系统添加安全性补丁,而不对系统的设计或体系结构做大的变更,这是不可能的。因此,添加补丁的办法既是低效的也是昂贵的。

在开展任何开发工作之前,迭代过程和增量过程的性质(第4章)使其难以解决所有的安全性问题。而且软件需求常常在开发过程中出现变更。此外,体系结构设计的决策可能对安全性的关注点产生直接影响。正因如此,在项目开始时,很难处理好所有的安全性问题。即使大部分安全问题预先已得到解决,软件过程后期的设计决策仍然会影响最终系统的安全性漏洞[Mei06]。

有效的软件过程包括一组合理的评审和调整措施。许多安全性活动都有互补作用,并且对软件质量具有协同效应。例如,众所周知,在测试之前,代码评审可以减少产品缺陷的数量,同时还可以消除潜在的安全性漏洞,从而提高软件的质量。

在制定计划时,项目预算和时间安排必须把安全性问题考虑在内,使得满足系统安全性目标^①的所需资源得到适当安排。作为安全性和保密性风险评估的一部分,每一项功能需求的要求都要检查,以便了解其是否会影响与系统安全性目标相关联的资产。在风险分析时,应确定和估计与每项损失相关联的成本。

确定处理系统特定威胁的机制常常延迟到将软件增量需求转化为其设计需求之后。这是因为在此情况下应该先确定攻击面。所谓攻击面(attack surface)是指存在于软件产品中的一组可获取并可利用的漏洞。许多安全性漏洞的交汇点将会被发现。例如,当其跨网络传播到数据库服务器时,在用户界面上以某种形式输入的信息可能被拦截。这样便可开发包括直接涉及攻击面的安全性规定在内的设计指南。

这可有助于区别安全性评审和一般的设计评审。侧重于安全性问题的代码评审应作为实现活动的一部分。应当根据系统设计活动中确定的相应安全性目标和威胁进行代码评审。

安全性测试是系统测试(第17章)的常规部分。安全性风险评估可以作为测试用例的来源,使得安全性测试更加受到关注。应急响应计划(Incident Response Plan, IRP)阐明了每个系统利益相关者要开展的活动,以响应特定的攻击[Pra07]。应急响应计划的充分评审应该是安全性验证过程的一个组成部分。

此外,验证应包括安全性操作和资产归档规程的评审。安全性风险管理计划应作为维护过程的一部分进行定期的评审。

在应用软件部署以后,报告出现安全性事件时,作为系统维护的一部分,开发人员应评

引述 世界上人类知识的总量大概每十年翻一番,我们的安全性知识只能靠学习能力的提高来得到。

Nathaniel
Branden

关键点 攻击面被定义为软件产品中一组可获取的和可利用的漏洞。

提问 什么是应急响应计划?

引述 当你认为自己能够处理所面临的任何事务时,那是因为你拥有世界为你必须提供的安全性。

Harry Browne

① 例如,客户数据的保护,与系统信息的机密性、完整性和可用性相关的法律和法规要求的认可,以及其他知识产权的保护等。

估安全性风险管理规程的有效性。如果系统变更规程（第 21 章）包括根本原因分析，这可能有助于发现在整个系统设计中的漏洞。

20.7 可信性系统验证

当我们在软件安全性的语境中考虑问题时，信任表明一个系统实体（或是一个组织）对另一系统实体（或组织）相信的程度。系统实体包含整个系统、子系统和软件构件。信任具有心理维度和技术维度。一般来说，假设第二个实体的表现正如第一个实体所期望的那样，第一个实体可以说信任第二个实体。论证这一假设的正确性是验证系统可信性的任务。虽然已经提出了多个信任模型 [Sin08]，但我们仍然关注确保系统在威胁模型中符合所提出的缓解风险的实际方法。

关键点 “信任”
表明一个系统实体（或组织）对另一实体（或组织）相信的程度。

验证工作能确保使用基于测试、审查和分析技术的特定和可量化度量，以评估可信性系统的需求 [She10]。测试度量包括检测到的故障次数与预测的故障次数之比，或已通过的安全性测试用例数与运行的总数之比。其他度量还包括正式评审活动的缺陷排除效率（第 23 章）。在分析活动中确保安全性测试用例回到已开发的安全性用例的可追溯性也是很有益的。

对于信任实体的所有协作者来说，用来证明安全性用例的证据必须是可以接受的和有说服力的。可信系统的用户应该确信系统没有可被利用的漏洞或恶意的逻辑。作为验证任务的结果，当遭到损害时，用户应在系统的可靠性和可存活性上充满信心。这便意味着对软件的损坏已降到最低限度，并且系统能够很快恢复到可接受的运行能力。具体的安全性测试用例和规程也是验证过程的一个重要组成部分 [Mea10]。

SafeHome | 安全性测试用例的生成

[场景] Vinod 的工作区。

[人物] Vinod Raman 和 Ed Robbins 都是软件团队成员。

[对话]

Vinod: 为了异地访问 SafeHome 系统的视频，我们需要提出安全性测试用例。

Ed: 我们应该从评审 Doug 和 Bridget（软件质量组组长）开发的安全性用例开始。

Vinod: 我认为可以让独立测试组（ITG）承包商做这项工作，但这似乎是非常简单的测试用例。同时为了回归测试，应当把它加到我们使用的测试用例组中。

Ed: 好的。用例密码要求用户登录到一个网站，使用一个有效的 ID 和两个密码，并且在请求视频按要求反馈后，用户要输入一个四位数字识别码。

Vinod: 这给了我们几个逻辑路径进行测试，

有四个用户输入的数据。每个输入都需用一个正常的值、一个不正确的值、一个空值和一个格式不正确的值进行测试。

Ed: 为覆盖所有的逻辑路径需要用 256 个不同的测试用例。

Vinod: 是的，的确是这样。我们还需要对每个用例的响应进行定义。

Ed: 基于安全性策略，针对每一条信息，用户可以尝试三次。

Vinod: 对，而且在每一次尝试失败后，提示用户输入数据。

Ed: 并且要在任何一条信息的第三次尝试中也失败，则系统应该发送电子邮件给公司和用户，以使他们警觉。

Vinod: 把测试用例以随机排序的方式提交给密码检查员可能是个好办法。我们可能要不只一次地运行我们的测试用例，确

信密码检查人员对历史记录并不敏感。

Ed: 我们应该写一个小程序, 运行所有这些测试用例并记录结果。

Vinod: 是的, 这其中有大量的工作。也许

我们应该让独立测试组 (ITG) 与 Bridget 的软件质量保证组一起来开展安全性测试。

如今, 软件质量测量已无法充分满足信任保证和安全性的要求。现有的测度 (例如, 考虑失效之间平均时间的可靠性 (reliability) 测度或是测量缺陷密度可信性 (dependability) 测度) 往往都忽略了很多因素, 这就可能使得软件受到损害, 并且容易受到攻击。在某种程度上这是因为许多度量并没有考虑到一个现实问题, 这就是存在一些活跃的不良人员在不断地挖掘软件漏洞。

在涉及信任的情况时, 基于实体过去的行为, 有效的安全性度量能够保持历史数据。例如, 当一个电子商务网站让它的买家和卖家作评价时, 就会考虑到所建立的信任。当然, 这类评级体系必须确保对被评价的实体有恰当且合理的确认, 而且有关实体并没有不准确的记录数据。这些问题有时会困扰信任报告系统。

美国国土安全部提倡采用安全软件设计的做法, 采用可靠而标准化的测量工具。理想的情况下, 这些开发工具的使用可以帮助开发者减少引入系统的漏洞数量 [Mea10]。这可能使那些被信任系统的用户在引导下做出有关系统可信性的决策。但正如系统可靠性那样, 用户也可能根据使用系统时所受到损失的程度做出判断。

软件工具 安全性工程

[目标] 安全性工程工具帮助人们在源代码中找到安全性漏洞。

[机制] 为了方便开发人员仔细检查, 通常让工具阅读源代码并标记编程结构来处理软件源代码。

[代表性工具]^①

- RATS (Rough Auditing Tool for Security) 由 Secure Software 开发 (<http://code.google.com/plrough-auditing-tool-for-security/>)。这是一个扫描工具, 它为安全性分析师提供了一组潜在的故障点, 以及问题的描述和建议的补救措施。
- ITS4 由 Cigital 开发 (<http://freecode.com/>

[projects/its4/](http://projects.its4.com/))。这是一个为寻找漏洞而对关键的安全性 C 源代码和 C++ 源代码作静态扫描的工具。

- SLAM 由 Microsoft 公司开发 (<http://research.microsoft.com/en-us/projects/slam/>)。该工具检查软件是否满足其使用界面的关键行为属性, 并且在设计界面和软件时帮助软件工程师确保实现可靠且运行安全。
- 许多安全性源代码的扫描工具可参看 <http://www.tech-fag.com/how-to-find-security-vulnerabilities-in-source-code.html>。

习题与思考题

20.1 考虑你自己的手机 App。首先简要地描述一个 App, 然后列出至少 3 ~ 5 个安全性风险。

20.2 针对上一题中提到的一个风险, 描述一个安全性缓解策略。

① 这里提到的工具只是此类工具的例子, 并不代表本书支持采用这些工具。在多数情况下, 工具的名字由各自的开发者注册为商标。

- 20.3 列出经常可能用于攻击 WebApp 的 5 种攻击模式。
- 20.4 描述应用于易趣 (eBay) 等投标网站上的信任模型。
- 20.5 为基于云的照片库描述安全性需求。
- 20.6 同源策略中哪些与可信赖系统有关?
- 20.7 对于个体消费者来说, 利用互联网确定单独发生身份被窃事件的年度平均成本。
- 20.8 在系统完成后, 若试图解决安全性风险, 请解释可能遇到的一些问题。
- 20.9 利用互联网找出用于生成网络钓鱼攻击模式的详细信息。
- 20.10 为在数据服务器上丢失的数据计算年度损失的预期值 (ALE), 其重建价值为 30 000 美元。因黑客的攻击每年数据损失为 5%, 而潜在损失达到 20 000 美元。

扩展阅读与信息资源

以下书籍都涉及重要的安全性问题: Vacca (《Computer and Information Security Handbook》, 2nd ed. Morgan Kaufman, 2013)、Goodrich 和 Tamassia (《Introduction to Computer Security》, Addison-Wesley, 2010)、Anderson (《Security Engineering》, 2nd ed., 2008)、Kern 等人 (《Foundations of Security》, Apress, 2007)、McGraw (《Software Security》, Addison-Wesley, 2006) 以及 Dowd 和他的同事 (《The Art of Software Assessment: Identifying and Preventing Software Vulnerabilities》, Addison-wesley, 2006)。Zalewski (《The Tangled Web: A Guide to Securing Modern Web Applications》, No Starch Press, 2011) 以及 Howard 和 LeBlanc (《Writing Secure Code》, 2nd ed., Microsoft Press, 2003) 的书都讨论了如何构建安全系统。Allen 等人 (《Software Security Engineering》, Addison-wesley, 2008) 提供了项目经理的观点、Howard 和 Lipner (《The Security Development Lifecycle》, Microsoft Press, 2006) 讨论了安全性工程过程以及 Schumacher 等人 (《Security Patterns》, Wiley, 2006) 对使用模式作为安全性工程的一个有效要素提出了他们的见解。

其他一些书籍侧重于系统“入侵”方面, 例如: Barnett 和 Grossan (《Web Application Defender's Cookbook: Battling Hackers Protecting Users》, Wiley, 2012)、Ottenheimer 和 Wallace (《Securing the Virtual Environment: How to Defend Against Enterprise Attack》, Wiley, 2012)、Studdard 和 Pinto (《The Web Application Hacker's Handbook》, Wiley, 2011)、Howard 和他的同事 (《24 Deadly Sins of Software Security》, WcGrau-Hill, 2009)、Erickson (《Hacking: The Art of Exploration》, No Starch Press, 2008)、Andrews 和 Whittaker (《How to Break Web Software》, Addison-wesley, 2006) 以及 Whittaker (《How to Break Software Security》, Addison-Wesley, 2003)。这些书都为软件工程师提供了有益的启示, 因为他们也需要从系统和应用软件是如何遭受攻击这个方面对安全性问题加深理解。

此外, Sikorski 和 Honig (《Practical Malware Analysis》, No Starch Press, 2012) 以及 Ligh 等人 (《Malware Analyst's Cookbook》, Wiley, 2010) 对不良软件的内部事务给出了出色的洞察。Swiderski (《Threat Modeling》, Microsoft Press, 2004) 对威胁建模做了详尽的讨论。

有些作者在书中给出了他们实施安全性测试的指导原则, 这些书包括: Allen (《Advanced Penetration Testing for Highly Secured Environments》, Packet Publishing, 2012)、O'Gorman (《Metasploit: The Penetration Tester's Guide》, No Starch Press, 2011)、Faircloth (《Penetration Tester's Open Source Tool Kit》, Syngress, 2011)、Engebretson (《The Basics of Hacking and Penetration Testing》, Syngress, 2011)、Faircloth (《Penetration Tester's Open Source Tool Kit》, Syngress, 2011)、Hope 和 Walther (《Web Security Testing Cookbook》, O'Reilly Media, 2008) 以及 Wysopal 等人 (《The Art of Software Security Testing》, Addison-Wesley, 2006)。

网上有各种安全性工程的信息可供利用。基于模式设计相关的最新参考文献可在 SEPA 网站 www.mhhe.com/pressman 找到。

软件配置管理

要点浏览

概念：开发计算机软件时会发生变更。因为变更的发生，所以需要有效地管理变更。软件配置管理（Software Configuration Management, SCM）也称为变更管理，是一组管理变更的活动。它通过下面的方式来管理变更：识别可能发生变更的工作产品，建立这些工作产品之间的关系，制定管理这些工作产品的不同版本的机制，控制所施加的变更，审核和报告所发生的变更。

人员：参与软件过程的每个人在某种程度上都参与变更管理，但是有时候也设专人来管理 SCM 过程。

重要性：如果你不控制变更，那么变更将控制你。这绝不是一件好事。一个未受控制的变更流可以很容易地将一个运行良好的软件项目带入混乱。结果会影响软件质量并且会推迟软件交付。为此，

变更管理是质量管理的重要部分。

步骤：在构建软件时会创建很多工作产品，因此每个工作产品都需要唯一标识。一旦成功完成标识，就可以建立版本和变更控制机制。为保证在变更发生时维护质量，变更过程需要审核；为了通知那些需要知道变更的人员，需要进行变更报告。

工作产品：软件配置管理计划（Software Configuration Management Plan）定义变更管理的项目策略。另外，当启动正式的 SCM 时，变更控制过程将产生软件变更请求、报告和工程变更工单。

质量保证措施：当每个工作产品都可以标识、跟踪和控制时，当每个变更都可以跟踪和分析时，当每个需要知道变更的人员都通知到时，你就做对了。

在开发计算机软件时，变更是不可避免的。而且，对于共同研发项目的软件开发团队来说，变更导致了混乱。如果变更之前没有经过分析，变更实现时没有做相应的记录，没有向需要了解变更的人员报告变更，或没有以一种能够改进质量并减少错误的方式控制变更，都会产生混乱。关于该问题，Babich[Bab86]是这样说的：

协调软件开发以最大限度地减少混乱的技术称为配置管理。配置管理是对软件开发团队正在构建的软件的修改进行标识、组织和控制的技术，其目标是使错误量减少到最小，并使生产率最高。

软件配置管理（SCM）是在整个软件过程中应用的一种普适性活动。变更可能随时出现，SCM 活动用于：（1）标识变更；（2）控制变更；（3）保证恰当地实施变更；（4）向其他可能的相关人员报告变更。

关键概念

基线

变更控制

配置审核

配置管理

配置对象元素

内容管理

标识

影响管理

中心存储库

SCM 过程

软件配置项

状态报告

版本控制

明确地区分软件支持和软件配置管理是很重要的。软件支持是一组发生在软件已经交付给客户并投入运行后的软件工程活动。而软件配置管理则是在软件项目开始时就启动,并且只有当软件被淘汰时才终止的一组跟踪和控制活动。

软件工程的主要目标是当发生变更时,使变更更容易地被接受,并减少变更发生时所花费的工作量。本章将探讨使得我们能够管理变更的具体活动。

21.1 软件配置管理概述

软件过程的输出信息可以分成三个主要类别:(1) 计算机程序(源代码和可执行程序);(2) 描述计算机程序的文档(针对不同的软件开发人员和客户);(3) 数据或内容(包含在程序内部和外部)。在软件过程中产生的所有信息项统称为软件配置。

随着软件工程工作的进行,软件配置项(Software Configuration Item, SCI)的层次结构就产生了。每一项可以是单个 UML 图一样小或者和完整的设计文档一样大的命名信息元素。如果每个 SCI 只是简单地产生了其他的 SCI,则几乎不会产生混乱。不幸的是,另一个变量进入过程中,就意味着变更。变更可以因为任意理由随时发生。事实上,正如系统工程第一定律(First Law of System Engineering) [Ber80] 所述:不管你处在系统生命周期的什么阶段,系统都可能发生变更,并且在整个生命周期中将会持续不断地提出变更的要求。

这些变更的起因是什么?这个问题的答案就像变更本身一样多变。然而,有 4 种基本的变更源:

- 新的业务或市场条件,引起产品需求或者业务规则的变更。
- 新的客户需要,要求修改信息系统产生的数据、产品提供的功能或基于计算机的系统提供的服务。
- 企业改组或扩大/缩小规模,导致项目优先级或软件工程团队结构的变更。
- 预算或进度的限制,导致系统或产品的重定义。

软件配置管理是一组用于在计算机软件的整个生命周期内管理变更的活动。SCM 可被视为应用于整个软件过程的软件质量保证活动。在下面的几节中,我们将描述能够帮助我们管理变更的主要 SCM 任务和重要概念。

21.1.1 SCM 场景^①

典型的配置管理(CM)工作场景包括:负责软件小组的项目经理、负责 CM 规程和方针的配置管理员、负责开发和维护软件产品的软件工程师以及使用软件产品的客户。在本场景中,我们假定由 6 个人组成的团队正在开发一个约 15000 行代码的小型软件。(注意,也可以组建更小或更大团队场景,但是,实际上每个这样的项目都面临着一个问题,就是 CM。)

在操作级别上,SCM 场景包括多种角色和任务。项目经理的职责是保证在确定的时间框架内开发出产品。因此,项目经理必须对软件的开发进展情况监控,找出问题,并对问题做出反应。这可通过建立和分析软

引述 除变更以外没有什么事情是永恒的。

Heraclitus,
公元前 500 年

提问 请求软件变更的起因是什么?

提问 每个参与变更管理的人员的职责和应从事的活动是什么?

① 本节摘自 [Dar01]。已经得到卡内基·梅隆大学软件工程研究所的同意,允许翻印由 Susan Dart [Dar01] 编写的“Spectrum of Functionality in CM Systems” (©2001 by Carnegie Mellon University)。

件系统状态报告并执行对系统的评审来完成。

配置管理员的职责不仅是保证代码的创建、变更和测试要遵循相应的规程和方针，还要使项目的相关信息容易得到。为了实现维护代码变更控制的技术，配置管理员可以引入正式的变更请求机制、变更评估机制（通过负责批准软件系统变更的变更控制委员会）和变更批准机制。配置管理员要为工程师们创建和分发任务单，并且还要创建项目的基本环境，而且，还要收集软件系统各个构件的统计信息，比如能够决定系统中哪些构件有问题的信息。

软件工程师的目标是高效地工作。也就是说，软件工程师在代码的创建和测试以及编写支持文档时不做不必要的相互交流；但同时，软件工程师们又要尽可能地进行有效的沟通和协调。特别是，软件工程师可以使用相应的工具来协助开发一致的软件产品；软件工程师之间可以通过相互通报任务要求和任务完成情况进行沟通和协调；通过合并文件，可以使变更在彼此的工作中传播。对于同时有多个变更的构件，要用某种机制来保证具有某种解决冲突和合并变更的方法。依据系统变更原因日志和究竟如何变更的记录，历史资料应该保持对系统中所有构件的演化过程的记录。软件工程师有他们自己创建、变更、测试和集成代码的工作空间。在特定点，可以将代码转变成基线，并从基线做进一步的开发，或生成针对其他目标机的变体。

关键点 一定存在某种机制，能够保证适当地跟踪、管理和执行同一个构件中同时发生的那些变更。

客户只是使用产品。由于产品处于 CM 控制之下，因此，客户要遵守请求变更和指出产品缺陷的正式规程。

理想情况下，在本场景中应用的 CM 系统应该支持所有的角色和任务。也就是说，角色决定了 CM 系统所需的功能。项目经理可以把 CM 看作是一个审核机制；配置管理员可以把 CM 看作控制、跟踪和制定方针的机制；软件工程师可以把 CM 看作变更、构建以及访问控制的机制；而用户可以把 CM 看作质量保证机制。

21.1.2 配置管理系统的元素

在 Susan Dart 关于软件配置管理的内容全面的白皮书 [Dar01] 中，她指明了开发软件配置管理系统时应该具备 4 个重要元素：

- **构件元素**是一组具有文件管理系统（如数据库）功能的工具，使我们能够访问和管理每个软件配置项。
- **过程元素**是一个动作和任务的集合，它为所有参与管理、开发和使用计算机软件的人员定义了变更管理（以及相关活动）的有效方法。
- **构建元素**是一组自动软件构建工具，用以确保装配了正确的有效构件（即正确的版本）集。
- **人员元素**是由实施有效 SCM 的软件团队使用的一组工具和过程特性（包括其他 CM 元素）。

以上这些元素（将在后面几节中详细讨论）并不是相互孤立的。例如，随着软件过程的演化，可能会同时用到构件元素和构建元素；过程元素可以指导多种与 SCM 相关的人员活动，因此也可以将其认为是人员元素。

21.1.3 基线

变更是软件开发中必然会出现的事情。客户希望修改需求，开发者希望修改技术方法，而管理者希望修改项目策略。为什么要修改呢？答案其实十分简单，随着时间的流逝，所有的软件参与者会得到更多的知识（关于他们需要什么、什么方法最好、如何既能完成任务又能赚钱），这些额外的知识是大多数变更发生的推动力，并且造成了很多软件工程实践者难以接受的事实——大多数变更是合理的！

基线是一个软件配置管理概念，它能够帮助我们在不严重阻碍合理变更的条件下控制变更。IEEE（IEEE 标准 610.12-1990）是这样定义基线的：

已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能修改它。

在软件配置项成为基线之前，可以较快地且非正规地进行变更。然而，一旦成为基线，虽然可以进行变更，但是必须应用特定的、正式的规程来评估和验证每次变更。

在软件工程范畴中，基线是软件开发中的里程碑，其标志是在正式技术评审中已经获得批准的一个或多个软件配置项的交付。例如，某设计模型的元素已经形成文档并通过评审，错误已被发现并得到纠正，一旦该模型的所有部分都经过了评审、纠正和批准，该设计模型就成为基线。任何对程序体系结构（在设计模型中已形成文档）的进一步变更只能在每次变更被评估和批准之后进行。虽然可以在任意细节层次上定义基线，但最常见的软件基线如图 21-1 所示。

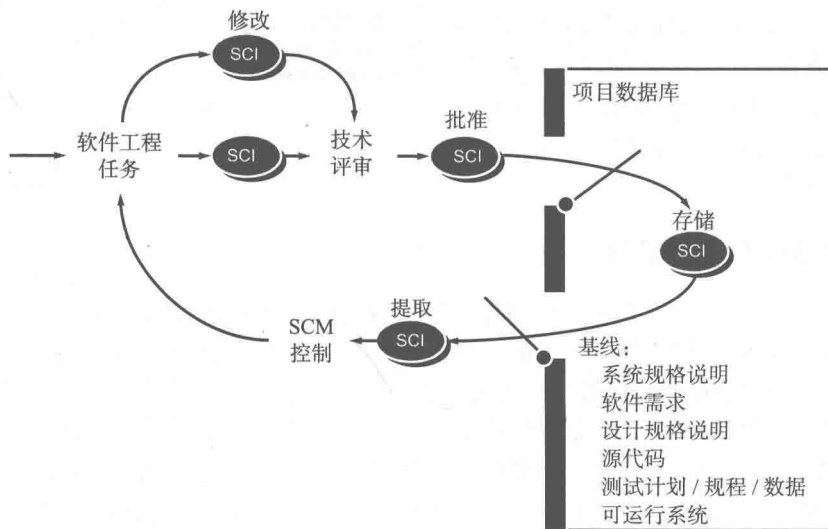


图 21-1 基线化的 SCI 和项目数据库

图 21-1 也说明了基线的形成过程。软件工程任务可能会产生一个或多个 SCI，在这些 SCI 经过评审并被批准之后，就可以将它们放置到项目数据库（也称为项目库或软件中心存储库，见 21.3 节）中。当软件团队中的成员想要修改某个基线 SCI 时，必须将该 SCI 从项目数据库中拷贝到工程师的私有工作区中。但是，这个被提取出的 SCI 只有在遵循 SCM 控制（在本章后面将讨论）的条件下才可以进行修改。图 21-1 中的箭头说明了某个已成为基线的 SCI 的修改路径。

建议 大多数软件变更是合理的，因此没有什么可抱怨的。更确切地说，要确信你已经有了恰当控制变更的机制。

建议 确保在一个集中的、可控的地点维护项目数据库。

21.1.4 软件配置项

软件配置项是在软件工程过程中创建的信息。在极端情况下，大型规格说明中的一节或大型测试用例集中的一个测试用例都可以看作一个 SCI。再实际点，一个 SCI 可以是工作产品的全部或部分（例如，一份文档、一整套测试用例，或者是一个已命名的程序构件）。

除了这些来自软件工作产品的 SCI 之外，很多软件工程组织也将软件工具列入配置管理的范畴，即特定版本的编辑器、编译器、浏览器以及其他自动化工具都被“固化”为软件配置的一部分。因为要使用这些工具来生成文档、源代码和数据，所以当要对软件配置进行变更时，必须得到这些工具。虽然问题并不多见，但一个工具的新版本（如编译器）有可能产生和原版本不同的结果。因此，就像它们协助开发的软件一样，工具也可以基线化为完整配置管理过程的一部分。

在现实中，是将 SCI 组织成配置对象，这些配置对象具有自己的名字，并且按类别存储在项目数据库中。配置对象具有一个名称和多个属性，并通过关系来表示与其他配置对象的“关联”。在图 21-2 中，分别定义了配置对象 DesignSpecification、DataModel、ComponentN、SourceCode 和 TestSpecification。各个对象之间的关系如图中箭头所示，单向箭头表示组合关系，即 DataModel 和 ComponentN 是 DesignSpecification 的组成部分。双向箭头说明对象之间的内在联系，如果 SourceCode 对象发生变更，软件工程师通过查看内在联系能够确定哪些对象（和 SCI）可能受到影响^①。

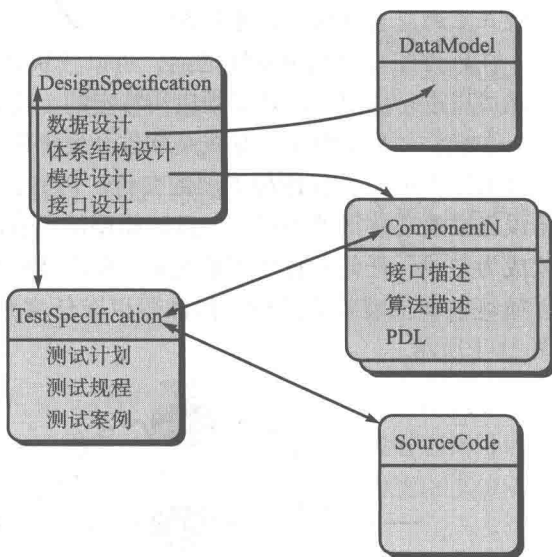


图 21-2 配置对象

21.1.5 依赖性和变更管理

我们介绍了可追溯性内容和可追溯矩阵的应用。可追溯矩阵是记录需求依赖性、架构决策（12.5 节）和缺陷原因（16.6 节）的一种方法。应当这些依赖性需求，当检验到变更时，它将会影响并指导测试用例的选择，还应将其用于回归测试（17.3.2 节）。依赖性管理被看作影响管理^②，这将帮助开发者关注如何改变他们的工作所带来的影响 [Sou08]。

影响分析既关注组织级行为，也关注个人活动，影响管理还包括两项补充内容：（1）确保软件开发者利用策略使他人对自己影响最小化；（2）鼓励软件开发者使用策略使自己对他人影响最小化。注意，最重要的是当一个开发者尝试将他的工作对其他人的影响最小化时，也会降低工作效率，同时其他人必须把对他的工作的影响最小化 [Sou08]。

最重要的是维护软件工作产品，确保开发者在 SCI 中觉察到了依赖性。当对 SCM 存储库进行检入 / 检出时，以及当批准如 21.2 节讨论的变更时，开发者必须建立原则。缺陷跟踪软件也很有用，可以帮助我们找到未发现的 SCI 依赖性。电子化沟通（E-mail、wikis、社交

① 这些关系可以在数据库内定义，数据库（中心存储库）的结构将在 21.3 节讨论。

② 影响管理将在 21.3.4 节讨论。

网络) 为开发者提供便利, 可以分享未记录的依赖性及其引发的问题。

21.2 SCM 中心存储库

SCM 中心存储库是一组机制和数据结构, 它使软件团队可以有效地管理变更。通过保证数据完整性、信息共享和数据集成, 它具有数据库管理系统的一般功能, 此外, SCM 中心存储库还为软件工具的集成提供了中枢, 它是软件过程流的核心。它能够使软件工程工作产品强制实施统一的结构和格式。

为了实现这些功能, 我们用术语元模型 (meta-model) 来定义中心存储库。元模型决定了在中心存储库中信息如何存储、如何通过工具访问数据、软件工程师如何查看数据、维护数据安全性和完整性的能力如何, 以及将现有模型扩展以适应新需求时的容易程度如何等。

21.2.1 一般特征和内容

中心存储库的特征和内容可以从两个方面来理解: 中心存储库存储什么, 以及中心存储库提供什么特定服务。中心存储库中存储的表示类型、文档和工作产品的详细分类如图 21-3 所示。

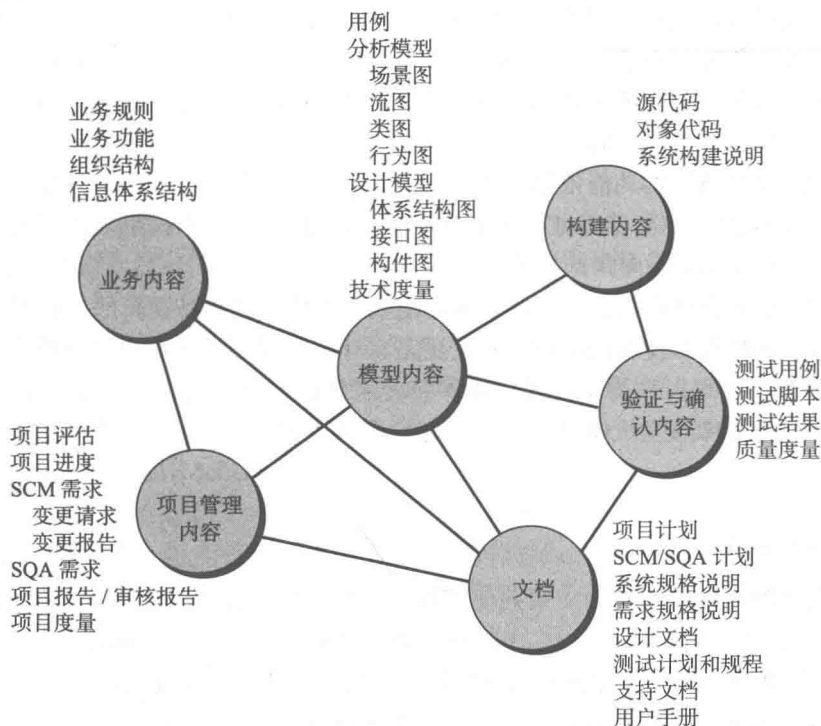


图 21-3 中心存储库的内容

一个健壮的中心存储库能够提供两种不同类型的服务: (1) 期望从任何一个复杂的数据库管理系统得到相同的服务类型; (2) 特定于软件工程环境的服务类型。

作为软件工程团队的中心存储库, 应该: (1) 集成或直接支持过程管理功能; (2) 支持在中心存储库中管理 SCM 功能的特定规则和维护数据; (3) 提供与其他软件工程工具的接口; (4) 能够存储各种数据对象 (例如,

网络资源 可以
通过网址 www.oracle.com/technology/products/repository/in-dex.html
获得商业化的中心存储库。

文本、图形、视频、音频)。

21.2.2 SCM 特征

为了支持 SCM, 中心存储库必须具有支持下列特征的工具集。

版本控制。随着项目的进展, 每个工作产品都可能有很多版本(21.3.2 节)。中心存储库必须能保存所有版本, 以便有效地管理产品发布, 并允许开发者在测试和调试过程中返回到早先的版本。

中心存储库必须能控制各种类型的对象, 包括文本、图形、位图、复杂文档及一些特殊对象, 如屏幕、报告定义、目标文件、测试数据和结果等。在成熟的中心存储库中可以按任意粒度跟踪对象版本, 例如, 可以跟踪单个数据定义或一组模块。

依赖性跟踪和变更管理。中心存储库要管理所存储的配置对象之间的各种关系。这些关系包括: 企业实体与过程之间的关系、应用系统设计各部分之间的关系、设计构件与企业信息体系结构之间的关系、设计元素与其他可交付工作产品之间的关系等。其中有些仅仅是关联关系, 而有些则是依赖关系或强制关系。

保持追踪这些关系的能力对中心存储库中存储信息的完整性以及基于中心存储库的可交付工作产品的生成是至关重要的, 而且这是中心存储库概念对软件开发过程改进最主要的贡献之一。例如, 如果修改了某 UML 类图, 则中心存储库能够检测出是否有相关联的类、接口描述和代码构件也需要进行修改, 并且能够提醒开发者哪些 SCI 受到了影响。

需求跟踪。这种特殊的功能依赖于相关管理, 并可以跟踪由特定需求规格说明产生的所有设计构件、架构构件以及可交付产品(正向跟踪)。此外, 还能够用来辨别指定的工作产品是由哪个需求产生的(反向跟踪)。

配置管理。配置管理设施能够跟踪表示特定项目里程碑或产品发布的一系列配置。

审核跟踪。审核跟踪使我们能够了解变更是在什么时候、由什么原因以及由谁完成等信息。变更的根源信息可以作为中心存储库中特定对象的属性进行存储。每当设计元素进行了修改时, 中心存储库触发机制有助于提示开发人员或创建审核信息条目(如变更理由)的工具。

21.3 SCM 过程

软件配置管理过程中定义的一系列任务具有 4 个主要目标: (1) 统一标识软件配置项; (2) 管理一个或多个软件配置项的变更; (3) 便于构建应用系统的不同版本; (4) 在配置随时间演化时, 确保能够保持软件质量。

能够取得上述 4 个目标的过程不应过于原则和抽象, 也不要太繁琐, 这个过程应该具有使软件团队能够解决一系列复杂问题的特色:

- 软件团队应该如何标识软件配置的离散元素?
- 组织应该如何管理程序(及其文档)的多个已有版本, 从而使变更能够高效地进行?
- 组织应该如何能在软件发布给客户之前和之后控制变更?
- 组织应该如何估算变更影响并有效地管理变更?
- 应该由谁负责批准变更并给变更确定优先级?

关键点 中心存储库必须能维护与软件的许多不同版本相关的 SCI。更重要的是, 它必须提供保证将这些 SCI 组装成一个具体版本配置的机制。

引述 任何变更, 甚至是为了实现更好产品的变更, 也伴随着缺陷和不适。

Arnold Bennett

提问 SCM 过程应该回答什么问题?

- 我们如何保证能够正确地完成变更?
- 应该采用什么机制去评价那些已经发生了的变更?

上述问题引导我们定义了 5 个 SCM 任务: 标识、变更控制、版本控制、配置审核和报告, 如图 21-4 所示。

图中, SCM 任务可以看作一个同心圆的层次结构。软件配置项 (SCI) 在它们的有效生存期内都要从内到外经历各个层次, 最终成为某应用程序或系统的一个或多个版本软件配置的组成部分。当一个 SCI 进入某层时, 该 SCM 过程层次所包含的活动可能适用, 也可能不适用。例如, 在创建一个新的 SCI 时, 必须对其进行标识, 但是, 如果对该 SCI 没有任何变更请求, 那就不必应用

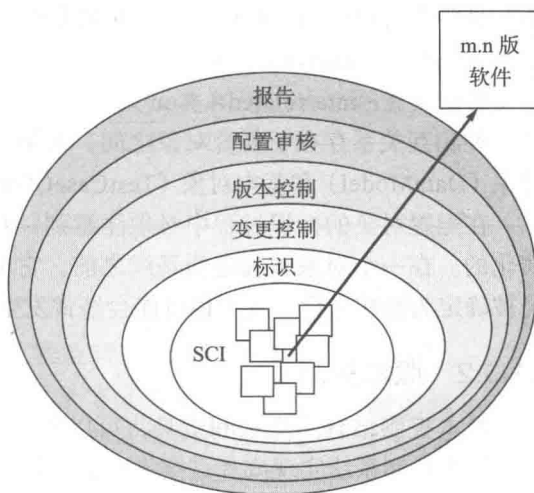


图 21-4 SCM 过程的层次

变更控制层的活动, 而直接将该 SCI 指定给软件的特定版本 (版本控制机制开始起作用了)。为了进行配置审核, 要维护 SCI 记录 (SCI 的名称、创建日期、版本标识等), 并将这些记录报告给那些需要知道的人员。下面, 我们将更详细地介绍每个 SCM 过程层次。

21.3.1 软件配置中的对象标识

为了控制和管理软件配置项, 必须对每个配置项单独命名, 然后用面向对象的方法进行组织。可以进行标识的对象有两种类型 [Cho89]: 基本对象和聚合对象[⊖]。基本对象是软件工程师在分析、设计、编码或测试过程中所创建的“信息单元”。例如, 一个基本对象可以是需求规格说明的一节、设计模型的一个部件、一个构件的源代码或用于测试代码的一组测试用例。聚合对象是基本对象和其他聚合对象的集合。例如, 图 27-2 中的 DesignSpecification 就是一个聚合对象。在概念上, 可将其视为一个已命名的 (已标识的) 指针列表, 指向诸如 ArchitectureModel 和 DataModel 的聚合对象, 以及诸如 ComponentN 和 UMLDiagramN 的基本对象。

每个对象都具有一组能够唯一地标识它的独特特征: 名称、描述、资源表及“实现”。对象名称是能够清楚地标识对象的一个字符串; 对象描述是一个数据项列表, 它能够标识出该对象所表示的 SCI 类型 (例如, 模型元素、程序、数据)、项目标识符以及变更和版本信息; 资源是“对象提供的、处理的、参考的实体, 或者是对象所需要的实体” [Cho89]。例如, 数据类型、特定功能甚至变量名都可以认为是对象资源。“实现”是一个指针, 对于基本对象, 该指针指向其文本单元, 对于聚合对象, 该指针为空。

标识配置对象时也可以考虑各个标识对象之间的关系。例如, 使用下面的简单表示:

类图 <part-of> 需求模型

需求模型 <part-of> 需求规格说明

可以为它们创建 SCI 层次结构。

关键点 针对配置对象建立的互联关系可用于评估变更的影响。

[⊖] 提出聚合对象的概念是为了将其作为描述软件配置的完整版本而提出的一种机制 [Gus89]。

在很多情况下，在对象层次结构中，对象是跨层次分支而互相关联的。这些交叉的结构关系可以用下面的方式表示：

数据模型 <interrelated> 数据流模型

数据模型 <interrelated> 类 m 的测试用例

第一种相互关系存在于复合对象之间，而第二种相互关系则是存在于聚合对象（DataModel）和基本对象（TestCaseClassM）之间。

在配置对象的标识过程中必须注意到，对象在整个软件过程中是不断演化的。在一个对象被确定为基线之前，它可能会变更很多次，甚至在已经被确定为基线之后，变更也可能会经常发生。

建议 即使项目数据库提供了建立这些关系的功能，但建立起来还是比较耗时，而且很难保持时效性。虽然这些关系对影响分析是很有用的，但是对于整个变更管理不是必需的。

21.3.2 版本控制

版本控制结合了规程和工具，可以用来管理在软件过程中所创建的配置对象的不同版本。版本控制系统实现或者直接集成了 4 个主要功能：（1）存储所有相关配置对象的项目数据库（中心存储库）；（2）存储配置对象所有版本（或能够通过与先前版本间的差异来构建任何一个版本）的版本管理功能；（3）使软件工程师能够收集所有相关配置对象和构造软件特定版本的制作功能；（4）版本控制和变更控制系统通常还有问题跟踪（也叫作错误跟踪）功能，使团队能够记录和跟踪与每个配置对象相关的重要问题的状态。

很多版本控制系统都可以建立变更集——构建软件特定版本所需要的所有变更（针对某些基线配置）的集合。Dart [Dar91] 写道：变更集“包含了对全部配置文件的所有变更、变更的理由、由谁完成的变更以及何时进行的变更等详细信息”。

可以为一个应用程序或系统标识很多已命名的变更集，这样就使软件工程师能够通过指定必须应用到基线配置的变更集（按名称）来构建软件的一个版本。为了实现这个功能，就要运用系统建模方法。系统模型包括：（1）一个模板，该模板包含构件的层次结构，以及构建系统时构件的“创建次序”；（2）构建规则；（3）验证规则^①。

在过去几年中，对于版本控制，人们已经提出了很多不同的自动化方法，这些方法的主要区别在于构建系统特定版本和变体属性时的复杂程度以及构建过程的机制。

软件工具 11 并发版本系统 (CVS)

通过工具来实现版本控制对于实现高效变更管理是必要的。并发版本系统（Concurrent Version System, CVS）是在版本控制中普遍使用的工具。它最初是为源代码设计的，但是可运用于任何文本文件。CVS 系统的功能有：（1）建立简单的中心存储库；（2）以一个命名文件来维护文件的所有版本，仅仅存储原始文件的各个渐进版本之间的差异；（3）为每位开发

者建立不同的工作目录以实现相互隔离，可避免同时对某个文件进行修改。当开发者完成各自的工作后，由 CVS 合并处理所有的修改。

要注意的是，CVS 不是一个“构建”系统，即它不能构建软件的特定版本。CVS 必须集成其他工具（例如 Makefile）才能完成这项工作。CVS 也不能实现变更控制过程（如变更请求、变更报告和错误

① 为了评估构件的变更将怎样影响其他构件，可能需要查询该系统模型。

跟踪)。

虽然有上述局限性,但 CVS 仍然“是一个优秀的、开源的、网络透明的版本控制系统,从单个开发者到大型的分散团队都可以使用”[CVS07]。CVS 的客户/服务器体系结构使用户可以从任何有

Internet 连接的地方访问文件,且开源理念使其适用于大部分流行的平台。

可以免费获得 Windows、Mac OS、Linux 和 UNIX 环境下的 CVS,应用的开源版本见 [CVS12]^①。

21.3.3 变更控制

James Bach[Bac98]很好地总结了现代软件工程范畴内变更控制的真实情况:

变更控制是至关重要的。但是,使变更控制至关重要的驱动力也使它令人厌烦。我们为变更所困扰,因为代码中一个极小的混乱就可能引起产品的失效;但是它也能够修复失效或具有奇妙的新能力。我们为变更所困扰,因为某个喜欢恶作剧的开发者可能破坏整个项目;但是,奇妙的想法也是源自那些喜欢恶作剧的人,而繁重的变更控制过程可能会严重阻碍他们进行创造性的工作。

引述 改进的艺术是在变更中保持有序,并且在有序中保持变更。

Alfred North
Whitehead

Bach 认为我们面临的是平衡问题。太多的变更控制会给我们带来问题,太少的变更控制又会给我们带来其他问题。

对于大型的软件工程项目,不受控制的变更会迅速导致混乱。对于这种大型项目,变更控制应该将人为制定的规程与自动工具结合起来。变更控制过程如图 21-5 所示,提交一个变更请求之后,要对其进行多方面的评估:技术指标、潜在副作用、对其他配置对象和系统功能的整体影响,以及变更的预计成本。评估的结果形成变更报告,由变更控制授权人(Change Control Authority, CCA,对变更的状态及优先级做出最终决策的人或小组)使用。对每个被批准的变更,需要建立工程变更工单(Engineering Change Order, ECO),ECO 描述了将要进行的变更、必须要考虑的约束以及评审和审核的标准。

关键点 应该注意到,许多变更请求可能联合起来形成一个 ECO,而 ECO 通常引起多个配置对象的变更。

可以将要进行变更的对象放到一个目录中,该目录只能由实施变更的软件工程师单独控制。完成变更之后,版本控制系统(如 CVS)可以更新原始文件。或者,可以将要进行变更的对象从项目数据库(中心存储库)中“检出”,进行变更,并应用适当的 SQA 活动,然后,再将对象“检入”到数据库,并应用适当的版本控制机制(21.3.2 节)构建该软件的下一个版本。

以上版本控制机制与变更控制过程集成在一起,实现了变更管理的两个主要元素——访问控制和同步控制。访问控制负责管理哪个软件工程师有权限访问和修改某个特定的配置对象;同步控制协助保证两个不同的人员完成的并行变更不会被相互覆盖。

你可能开始对图 21-5 描述的变更控制过程所蕴含的繁文缛节感到不适,这种感觉是很正常的。没有适当的防护措施,变更控制可能会阻碍进展,也可能产生不必要的繁琐手续。大多数拥有变更控制机制的软件开发者(不幸的是,很多人没有)通过许多控制环节来避免上面提到的这些问题。

^① 从 <http://olex.openlogic.com/packages/cvs> 下载 [CVS12]。



图 21-5 变更控制过程

在 SCI 成为基线之前，只需要进行非正式的变更控制。还在讨论之中的配置对象（SCI）的开发者可以进行任何变更，只要项目和技术需求证明这些变更是适当的（只要变更不会影响到开发者工作范围之外的系统需求）。一旦配置对象经过正式技术评审并被批准，它就会成为基线^①。一旦 SCI 成为基线，就可以实现项目级变更控制了。这时，若要进行变更，开发者必须得到项目管理者的批准（如果变更是“局部的”），如果该变更影响到其他 SCI，则必须得到 CCA 的批准。在某些情况下，开发者无需生成正式的变更请求、变更报告和 ECO，但是，必须对每个变更进行评估，并对所有的变更进行跟踪和评审。

当交付软件产品给客户时，正式的变更控制就开始实施了，正式的变更控制规程如

建议 多选择一些变更控制，即使你认为不需要这么多。很可能“较多”就是正好的。

① 也可能因为其他理由创建基线。例如，当“每日构建”基线创建后，在给定时间提交的所有构件都变成下一日工作的基线。

图 21-5 所示。

CCA 在控制的第二层和第三层中扮演着主动的角色。CCA 可以是一个人 (项目经理), 也可以是很多人 (例如, 来自软件、硬件、数据库工程、支持、市场等方面的代表), 这取决于软件项目的规模和性质。CCA 的作用是从全局的观点来评估变更对 SCI 之外事物的影响。变更将对硬件产生什么影响? 变更将对性能产生什么影响? 变更将怎样改变客户对产品的感觉? 变更将对产品的质量和可靠性产生什么影响? 还有很多其他问题都需要 CCA 来处理。

引述 变更是不可避免的, 自动贩卖机除外。

Bumper sticker

SafeHome SCM 问题

[场景] Doug Miller 的办公室, SafeHome 软件项目开始之初。

[人物] Doug Miller (SafeHome 软件团队经理)、Vinod Raman、Jamie Lazar 以及产品软件工程团队的其他成员。

[对话]

Doug: 我知道时间还早, 但是我们应该开始讨论变更管理了。

Vinod (笑着说): 很难。今天早上市场部打电话来, 有几个需要“重新考虑”的地方。都不是主要的, 但这只是刚刚开始。

Jamie: 在以前的项目中, 我们的变更管理就非常不正式。

Doug: 我知道, 但是这个项目规模更大、更显眼, 使我想起……

Vinod (不断点头): 我们已经被“家庭照明控制”项目中不受控制的变更折腾得要命……想起延期就……

Doug (皱着眉): 我不愿意再经历这样的恶梦。

Jamie: 那我们该做什么?

Doug: 在我看来, 应该做三件事。第一件, 我们必须建立 (或借用) 一个变更控制过程。

Jamie: 你的意思是如何请求变更吗?

Vinod: 是的。但也包括如何评估变更, 如何决定何时进行变更 (如果由我们决定), 以及如何记录变更所产生的影响。

Doug: 第二件, 我们必须获得一个适用于变更控制和版本控制的 SCM 工具。

Jamie: 我们可以为所有的工作产品创建一个数据库。

Vinod: 在这里叫 SCI, 绝大部分好用的工具都不同程度地支持这个功能。

Doug: 这是一个良好的开端, 现在我们必须……

Jamie: 嗯, Doug, 你说过是三件事的……

Doug (笑着说): 第三件, 我们必须使用工具, 而且无论如何大家都要遵守变更管理过程。行吗?

21.3.4 影响管理

每次变更发生时, 我们都必须考虑软件工作产品的依赖性网络, 影响管理 (impact management) 涉及正确理解这些依赖关系, 以及控制它们对其他软件配置项 (及其负责人) 的影响。

影响管理依靠三种动作来实现 [Sou08]。首先, 影响网络识别出软件团队 (及其他利益相关者) 中可能引发变更或受到软件变更影响的人员数目。软件体系结构的清晰定义 (第 12 章) 对于生成影响网络很有帮助。其次, 正向影响管理 (forward impact management) 评估自身变更对影响网络上的成员的影响, 然后告知受到变更影响的成员。最后, 反向影响管理

(back impact management) 检查其他团队成员所做的变更及其对自己工作的影响，从而体现减轻影响的机制。

21.3.5 配置审核

标识、版本控制和变更控制帮助软件开发者维持秩序，否则情况可能将是混乱和不断变化的。然而，即使最优秀的控制机制也只能在 ECO 建立之后才可以跟踪变更。我们如何能够保证变更的实现是正确的呢？答案分两个方面：（1）技术评审；（2）软件配置审核。

技术评审关注的是配置对象在修改后的技术正确性。评审者要评估 SCI，以确定它与其他 SCI 是否一致、是否有遗漏或是否具有潜在的副作用。除了那些非常微不足道的变更之外，应该对所有变更进行技术评审。

作为技术评审的补充，软件配置审核针对在评审期间通常不被考虑的特征对配置对象进行评估。软件配置审核要解决以下问题：

- 1. 在 ECO 中指定的变更已经完成了吗？引起任何额外的修改了吗？
- 2. 是否已经进行了技术评审来评估技术正确性？
- 3. 是否遵循了软件过程，是否正确地应用了软件工程标准？
- 4. 在 SCI 中“显著标明”所做的变更了吗？是否说明了变更日期和变更者？配置对象的属性反映出该变更了吗？
- 5. 是否遵循了 SCM 规程中标注变更、记录变更和报告变更的规程？
- 6. 是否已经正确地更新了所有相关的 SCI？

提问 在配置审核期间需要关注的主要问题是什
么？

在某些情况下，这些审核问题将作为技术评审的一部分。但是，当 SCM 是正式活动时，配置审核将由质量保证小组单独进行。这种正式的配置审核还能够保证将正确的 SCI（按版本）集成到特定的版本构建中，并且能够保证所有文档都是最新的，且与所构建的版本是一致的。

21.3.6 状态报告

配置状态报告（Configuration Status Reporting, CSR，有时称为状态账目）是一项 SCM 任务，它解答下列问题：（1）发生了什么事？（2）谁做的？（3）什么时候发生的？（4）会影响其他哪些事情？

配置状态报告的信息流如图 21-5 所示，每当赋予 SCI 新的标识或更改其标识时，就会产生一个 CSR 条目；每当 CCA 批准一个变更（即创建一个 ECO）时，就会产生一个 CSR 条目；每当进行配置审核时，其结果要作为 CSR 任务的一部分提出报告。CSR 的结果可以放置到一个联机数据库中或 Web 站点上，以便软件开发者或维护人员可以按照关键词分类来访问变更信息。此外，定期生成的 CSR 报告使管理者和开发人员可以评估重要的变更。

建议 为每个配置对象创建一个“须知”列表，并实时更新。当变更发生时，保证列表上的每个人都被通知到。

软件工具 | SCM 支持

[目标] SCM 工具可以支持 21.3 节所讨论的一个或多个过程活动。

[机制] 大部分最新的 SCM 工具都与一个

中心存储库（一个数据库系统）相连，能够提供标识、版本控制和变更控制、审核及报告功能。

[代表性工具]^①

- Software Change Manager。由 Computer Associates (<http://www.ca.com/us/products/Detail/ca-change-manager-enterprise-workbench.aspx>) 发行, 是一个多平台 SCM 系统。
 - ClearCase。由 Rational 开发, 提供 SCM 的一系列功能 (<http://www-03.ibm.com/software/products/us/en/clearcase>)。
 - Serena Dimensions CMF。由 Serena (<http://www.serena.com/US/products/zmf/index.aspx>) 发行, 提供了适用于传统软件和 WebApp 的全套 SCM 工具。
 - Allura。由 SourceForge (<http://sourceforge.net/p/allura/wiki/Allura%20Wiki>) 发行, 提供了版本控制、构建功能、问题/错误跟踪及许多其他管理功能。
 - SurroundSCM。由 Seapine Software (www.seapine.com) 开发, 提供完整的变更管理功能。
 - Vesta。由 Compac (www.vestasys.org) 发行, 是一个能够支持小型 (<10 KLOC) 和大型 (10 000 KLOC) 项目的没有版权限制的 SCM 系统。
- 商用 SCM 工具与环境的综合列表可以在 www.cmtoday.com 找到。

习题与思考题

- 21.1 为什么“系统工程第一定律”会成立? 变更的主要理由有 4 个, 对每个理由都举出几个特例。
- 21.2 实现高效 SCM 系统必需的 4 个元素是什么? 简要介绍每个元素。
- 21.3 用自己的话谈谈定义基线的理由。
- 21.4 假定你是某个小项目的负责人, 你会为项目定义什么基线? 如何控制它们?
- 21.5 设计一个项目数据库 (中心存储库) 系统, 使软件工程师能够存储、交叉引用、跟踪、更新和变更所有重要的软件配置项。数据库应该如何处理同一程序的不同版本? 源代码的处理会与文档的处理有所不同吗? 两个开发者应该如何避免同时对同一个 SCI 执行不同的修改?
- 21.6 研究某现有的 SCM 工具, 然后大概描述它是如何实现版本控制和配置对象控制的。
- 21.7 关系〈 part-of 〉和〈 interrelated 〉表示配置对象之间的简单关系, 描述 5 种可能在 SCM 中心存储库中用到的其他关系。
- 21.8 研究某现有的 SCM 工具, 并描述它实现版本控制的方法。此外, 阅读 2~3 篇有关 SCM 的文章, 并描述用于版本控制的不同数据结构和引用机制。
- 21.9 设计一个用在配置审核中的检查表。
- 21.10 SCM 审核和技术评审有什么区别? 它们的作用可以归纳为一种评审吗? 请说明正反两方面的观点。
- 21.11 简要描述传统软件的 SCM 与 WebApp 的 SCM 之间有何不同。
- 21.12 什么是内容管理? 利用网络资源研究内容管理工具的特性, 并给出简要的总结。

扩展阅读与信息资源

关于 SCM 的最新资料包括 Aiello 和他的同事 (《 Configuration Management Best Practices: Practical Methods That Work in the Real World 》, Addison-Wesley, 2010)、Moreira (《 Adapting configuration Management for Agile Teams: Balancing Sustainability and Speed 》, Wiley, 2009)、Duvall 和他的同事 (《 Continuous Integration: Improving Software Quality and Reducing Risk 》, Addison-Wesley, 2007)、Leon (《 Software Configuration Management Handbook 》, second edition, Artech House Publishers, 2005)、Moreira (《 The Build Master: Microsoft Software Configuration Management Best

Practice 》, Addison Wiley, 2005)、Keyes (《Software Configuration Management 》,Auerbach, 2004) 以及 Hass (《Configuration Management Principles and Practice 》, Addison-Wesley, 2002), 每本书都非常详细地描述了整个 SCM 过程。Moraia (《Software Configuration Management Implementation Roadmap 》, Wiley, 2004) 提出了独特的指南, 用以帮助那些在组织内必须实施 SCM 的工程人员。Lyon (《Practical CM III: Best Practices for the 21st Century 》, Raven Publishing, 2013, www.configuration.org) 为 CM 专业人员写了一本内容全面的指导书, 书中包含了实现配置管理系统的全方位的实用指导原则 (每年更新)。Girod 和 Shpichko (《IBM Rational ClearCase 7.0: Master the Tools That Monitor, Analyze, and Manage Software Configurations 》, Packt, 2011)、Bellagio 和 Mulligan 以一个或多个当今流行的 SCM 工具为例介绍了 SCM。

Berczuk 和 Appleton (《Software Configuration Management Patterns 》, Addison-Wesley, 2003) 介绍了有助于理解 SCM 和实现高效 SCM 系统的很多有用模式。Brown 等人 (《Anti-Patterns and Patterns in Software Configuration Management 》, Wiley, 1999) 叙述了在 SCM 过程中不能做的那些事情 (反模式), 然后找出它们的解决办法。Humble 和 Fowler (《Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation 》, Addison-Wesley, 2010) 以及 Bays (《Software Release Methodology 》, Prentice-Hall, 1999) 重点讲述成功产品的发布机制, 这也是对有效 SCM 的重要补充。

由于 WebApp 的动态性, 使内容管理成为 Web 工程师关注的主题。Rockley 和 Cooper (《Managing Enterprise Content: A Unified Content Strategy 》, 2nd ed., New Riders, 2012)、Jenkins 和他的同事 (《Managing Content in the Cloud-Enterprise Content Management 2.0 》, Open Text Corporation, 2010 和《Managing Content in the Cloud-Enterprise Content Management 》, Open Text Corporation, 2005)、White (《The Content Management Handbook 》, Curtin University Books, 2005)、Jenkins 和他的同事 (《Enterprise Content Management Methods 》, Open Text Cooperation, 2005)、Bioko[Bio04]、Mauthe 和 Thomas (《Professional Content Management Systems 》, Wiley, 2004)、Addey 和他的同事 (《Content Management Systems 》, Glasshaus, 2003)、Rockly (《Managing Enterprise Content 》, New Riders, 2002)、Hackos (《Content Management for Dynamic Web Delivery 》, Wiley, 2002) 以及 Nakano (《Web Content Management 》, Addison-Wesley, 2001) 介绍了这方面的有价值的内容管理方法。

除了软件配置管理的一般主题外, Halvorson 和 Bach (《Content Strategy for the Web 》, 2nd ed., New Riders, 2012)、Hauschildt (《CMS Made Simple 1.6: Beginners' Guide 》, Packt, 2010)、Lim 和他的同事 (《Enhance Microsoft Content Management Server with ASP.Net 2.0 》, Packt Publisher, 2006)、Ferguson (《Creating Content Management Systems in Java 》, Charles River Media, 2006)、IBM Redbook (《IBM Workplace Web Content Management for Portal 5.1 》和《IBM Workplace Web Content Management 2.5 》, Vivante, 2006)、Fritz 和他的同事 (《Typo3: Enterprise Content Management 》, Packt Publishing, 2005) 以及 Forta (《Reality ColdFusion: Intranets and Content Management 》, Pearson Education, 2002) 也描述了在使用具体工具和语言环境下的内容管理。

网上有大量关于软件配置管理和内容管理的信息资源。有关软件配置管理的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 的 “software engineering resources” 下找到。

管理软件项目

在本书的这一部分，将学习计划、组织和监控软件项目所需要的管理技术。

在下面的章节中，我们将讨论以下问题：

- 在软件项目进行期间，为什么要对人员、过程和问题进行管理？
- 如何使用软件度量来管理软件项目和软件过程？
- 软件团队如何可靠地估算软件项目的工作量、成本和工期？
- 采用什么技术来正式评估影响项目成功的风险？
- 软件项目经理如何选择软件工程工作任务集？
- 如何编制项目进度计划？

回答了这些问题后，就为管理软件项目做好了较充分的准备，便可以在某种程度上按时交付高质量的产品。

项目管理概念

要点浏览

概念: 虽然很多人在悲观的时候接受 Dilbert 的“管理”观点,但是在构建基于计算机的系统和产品时管理仍然是一项非常必要的活动。在软件从初始概念演化为可供有效使用的产品的过程中,项目管理涉及对人员、过程和所发生事件的计划和监控。

人员: 在软件项目中,每个人或多或少都做着“管理”工作。但是,管理活动的范围各不相同。软件工程师管理他们的日常活动,计划和监控技术任务。项目经理计划和监控软件工程师团队的工作。高级管理者协调业务和软件专业人员之间的关系。

重要性: 构建计算机软件是一项复杂的任务,尤其是当它涉及很多人员长期共同工作的时候。这就是为什么软件项目需要管理的原因。

步骤: 理解 4P——人员 (People)、产

品 (Product)、过程 (Process) 和项目 (Project)。必须将人员组织起来以有效地完成软件工作。必须和客户及其他利益相关者很好地沟通,以便了解产品的范围和需求;必须选择适合于人员和产品的过程;必须估算完成工作任务的工作量和工作时间,从而制定项目计划,包括定义工作产品、建立质量检查点以及确定一些机制来监控计划所规定的工作。

工作产品: 在管理活动开始时,首先是制定项目计划。该计划定义将要进行的过程和任务,安排工作人员,确定评估风险、控制变更和评价质量的机制。

质量保证措施: 在按时并在预算内交付高质量的产品之前,你不可能完全肯定项目计划是正确的。不过,作为项目经理,鼓励软件人员协同工作以形成一支高效的团队,并将他们的注意力集中到客户需求和产品质量上,这肯定是正确的。

Meiler Page-Jones [Pag85] 在其关于软件项目管理论著的序言中给出了以下一段陈述,这引起了许多软件工程顾问的共鸣:

我拜访了很多商业公司——好的和不好的,我又观察了很多 (IT) 管理者的业绩——好的和不好的。我常常恐惧地看到,这些管理者徒劳地与恶梦般的项目斗争着,在根本不可能完成的最后期限的压力下苦苦挣扎,或者是在交付了用户极为不满意的系统之后,又继续花费大量的时间去维护它。

Page-Jones 所描述的正是源于一系列管理和技术问题的症状。不过,如果在事后再剖析一下每个项目,很有可能发现一个共同的问题:项目管理太弱。

在本章以及第 23 ~ 26 章中,将给出进行有效的软件项目管理的关键

关键概念

敏捷团队
协调和沟通
关键实践
人员
问题分解
产品
项目
软件范围
软件团队
利益相关者
团队负责人
W^SHH 原则

概念。本章考虑软件项目管理的基本概念和原则。第23章介绍过程和项目度量，这是做出有效管理决策的基础。第24章讨论用于估算成本的技术。第25章将帮助你编制实际的项目进度表。第26章阐述了进行有效的风险监测、风险缓解和风险控制的管理活动。

22.1 管理涉及的范围

有效的软件项目管理集中于4P，即人员、产品、过程和项目，它们的顺序不是任意的。任何管理者如果忘记了软件工程工作是人的智力密集的劳动，他就永远不可能在项目管理上取得成功；任何管理者如果在项目开发早期没有鼓励利益相关者之间的广泛交流，他就冒着为错误的问题构建了“良好的”解决方案的风险；对过程不在意的管理者可能冒着把有效的技术方法和工具插入真空中的风险；没有建立可靠的项目计划就开始工作的管理者将危及产品的成功。

22.1.1 人员

从20世纪60年代起人们就一直在讨论要培养有创造力、高技术水平的软件人员。实际上，“人的因素”的确非常重要，美国卡内基·梅隆大学的软件工程研究所（SEI）认识到这一事实——“每个组织都需要不断地提高他们的能力来吸引、发展、激励、组织和留住那些为实现其战略业务目标所需的劳动力”[Cur01]，并开发了一个人员能力成熟度模型（People-CMM）。

人员能力成熟度模型中针对软件人员定义了以下关键实践域：人员配备、沟通与协调、工作环境、业绩管理、培训、报酬、能力素质分析与开发、个人事业发展、工作组发展以及团队精神或企业文化培养等。People-CMM成熟度达到较高水平的组织，更有可能实现有效的软件项目管理实践。

People-CMM与软件能力成熟度集成模型相伴而生，后者可指导组织创建一个成熟的软件过程。与软件项目的人员管理及人员结构相关的问题将在本章后面的内容中考虑。

22.1.2 产品

在制定项目计划之前，应该首先确定产品的目标和范围，考虑可选的解决方案，识别技术和管理上的限制。如果没有这些信息，就不可能进行合理的（精确的）成本估算，也不可能进行有效的风险评估和适当的项目任务划分，更不可能制定可管理的项目进度计划来给出意义明确的项目进展标志。

作为软件开发者，必须与其他利益相关者一同定义产品的目标和范围。在很多情况下，这项活动是作为系统工程或业务过程工程的一部分开始的，并一直持续到作为软件需求工程的第一步（第7章）。确定产品的目标只是识别出产品的总体目标（从利益相关者的角度），而不用考虑如何实现这些目标。而确定产品的范围，是要标识出产品的主要数据、功能和行为特性，而且更为重要的是，应以量化的方式界定这些特性。

了解产品的目标和范围之后，就要开始考虑备选的解决方案。虽然这一步并不讨论细节，但可以使管理者和参与开发的人员根据给定的约束条件选择“最好”的方案，约束条件

建议 坚持敏捷过程方法（第5章）的人指出敏捷过程比其他过程更简单，这可能是真的。但是敏捷过程仍然有一个过程，敏捷软件工程依然需要规则。

包括产品交付期限、预算限制、可用人员、技术接口以及其他各种因素。

22.1.3 过程

软件过程(第3~5章)提供了一个框架,在该框架下可以制定软件开发的综合计划。一小部分框架活动适用于所有软件项目,不用考虑其规模和复杂性。多种不同的任务集合(每一种集合都由任务、里程碑、工作产品以及质量保证点组成)使得框架活动适合于不同软件项目的特性和项目团队的需求。最后是普适性活动——如软件质量保证、软件配置管理、测量,这些活动覆盖了过程模型。普适性活动独立于任何一个框架活动,且贯穿于整个过程之中。

22.1.4 项目

我们实施有计划的、可控制的软件项目的主要理由是:这是我们知道的管理复杂事物的唯一方法。然而,软件团队仍需要努力。在对1998~2004年的250个大型软件项目的一份研究中,Capers Jones[Jon04]发现,“大约有25个项目被认为是成功的,达到了他们的计划、成本和质量目标;大约有50个项目延迟或超期在35%以下;而大约有175个项目经历了严重的延迟和超期,或者没有完成就中途夭折了。”虽然现在软件项目的成功率可能已经有所提高,但项目的失败率仍然大大高于它的应有值^①。

为了避免项目失败,软件项目经理和开发产品的软件工程师必须避免一些常见的警告信号,了解实施成功的项目管理的关键因素,还要确定计划和监控项目的一目了然的方法。这些问题将在22.5节及以后的章节中讨论。

引述 一个项目如同一次公路旅行,有些项目是简单的、常规性的,就像在白天开车去购物。但是值得做的大多数项目就像在夜晚驾驶一辆卡车离开公路驶入大山中一样。
Cem Kaner,
James Bach,
Bret Pettichord

22.2 人员

人们开发计算机软件,并取得项目的成功,是由于他们受过良好的训练并得到了激励。我们所有人,从高级工程副总裁到基层开发人员,常常认为人员是不成问题的。虽然管理者常常表态说人员是最重要的,但有时他们言行并不一致。本节将分析参与软件过程的利益相关者,并研究组织人员的方式,以实现有效的软件工程。

22.2.1 利益相关者

参与软件过程(及每一个软件项目)的利益相关者可以分为以下5类:

- 高级管理者负责定义业务问题,这些问题往往对项目产生很大影响。
- 项目(技术)管理者必须计划、激励、组织和控制软件开发人员。
- 开发人员拥有开发产品或应用软件所需的技能。
- 客户阐明待开发软件的需求,包括关心项目成败的其他利益相关者。
- 最终用户是软件发布成为产品后直接与软件进行交互的人。

① 看到这些统计数据,人们自然会问计算机的影响又为何持续呈指数增长。我们认为,部分原因是:相当数量的“失败”项目在刚开始时就是构想拙劣的,客户很快就失去了兴趣(因为他们所需要的并不像他们最初想象的那样重要),进而取消了这些项目。

每个软件项目都有上述人员的参与^①。为了获得高效率，项目团队必须以能够最大限度地发挥每个人的技术和能力的方式进行组织，这是团队负责人的任务。

22.2.2 团队负责人

项目管理是人员密集型活动，因此，胜任开发的人却常常有可能是拙劣的团队负责人，他们完全不具备管理人员的技能。正如 Edgemon 所说：“很不幸但却经常是这样，人们似乎碰巧落在项目经理的位置上，也就意外地成为项目经理”。[Edg95]

在一本关于技术领导能力的优秀论著中，Jerry Weinberg[Wei86] 提出了领导能力的 MOI 模型（Motivation（激励）、Organization（组织）、Ideas or Innovation（思想或创新））。

提问 当我们选择软件项目的负责人时，我们在寻找什么？

激励：（通过“推”或“拉”）鼓励技术人员发挥其最大才能的一种能力。

组织：形成能够将最初概念转换成最终产品的现有过程（或创造新的过程）的能力。

思想或创新：即使必须在特定软件产品或应用系统的约束下工作，也能鼓励人们去创造并让人感到有创造性的一种能力。

Weinberg 提出，成功的项目负责人应采用一种解决问题的管理风格。也就是说，软件项目经理应该注重理解要解决的问题，把握住涌现的各种意见，同时让项目团队的每个人都知道（通过言语，更重要的是通过行动）质量很重要，不能妥协。

关于一个具有实战能力的项目经理应该具有什么特点，另一种观点 [Edg95] 则强调了以下 4 种关键品质。

解决问题。具有实战能力的软件项目经理能够准确地诊断出最为密切相关的技术问题和组织问题；能够系统地制定解决方案，适当地激励其他开发人员来实现该方案；能够将在过去项目中学到的经验应用到新环境中；如果最初的解决方案没有结果，能够灵活地改变方向。

引述 用最简单的话来说，负责人是这样的人，他知道自己想去哪里，并起身朝那里走。

John Erskine

管理者的特性。优秀的项目经理必须能够掌管整个项目。必要的时候要有信心进行项目控制，同时还要允许优秀的技术人员按照他们的本意行事。

成就。为了优化项目团队的生产效率，一位称职的项目经理必须奖励那些工作积极主动并且做出成绩的人。必须通过自己的行为表明出现可控风险并不会受到惩罚。

影响和队伍建设。具有实战能力的项目经理必须能够“理解”人。他必须能理解语言和非语言的信号，并对发出这些信号的人的要求做出反应。项目经理必须能在高压力的环境下保持良好的控制能力。

22.2.3 软件团队

几乎可以说有多少开发软件的组织，就有多少种软件开发人员的组织结构。不管怎么说，组织结构不能轻易改变。至于组织改变所产生的实际的和行政上的影响，并不在软件项目经理的责任范围内。但是，对新的软件项目中所直接涉及的人员进行组织，则是项目经理的职责。

引述 并非每个小组都是团队，并非每个团队都是有有效的。

Glenn Parker

“最好的”团队结构取决于组织的管理风格、团队里的人员数目与技

① 开发 WebApp 或移动 App 时，在内容创作方面需要其他非技术人员参与。

能水平，以及问题的总体难易程度。Mantei[Man81]提出了规划软件工程团队结构时应该考虑的7个项目因素：（1）待解决问题的难度；（2）开发程序的规模，以代码行或者功能点来度量；（3）团队成员需要共同工作的时间（团队生存期）；（4）能够对问题做模块化划分的程度；（5）待开发系统的质量要求和可靠性要求；（6）交付日期的严格程度；（7）项目所需要的友好交流的程度。

Constantine[Con93]提出了软件工程团队的4种“组织范型”：

1. 封闭式范型。按照传统的权利层次来组织团队。当开发与过去已经做过的产品相似的软件时，这种团队十分有效。但在这种封闭式范型下难以进行创新性的工作。
2. 随机式范型。松散地组织团队，团队工作依赖于团队成员个人的主动性。当需要创新或技术上的突破时，按照这种随机式范型组织的团队很有优势。但当需要“有次序地执行”才能完成工作时，这种团队就会陷入困境。
3. 开放式范型。试图以一种既具有封闭式范型的控制性，又包含随机式范型的创新性的方式来组织团队。工作是大家相互协作完成的。良好的沟通和根据团队整体意见做出决策是开放式范型的特征。开放式范型的团队结构特别适合于解决复杂的问题，但可能不像其他类型的团队那么有效率。
4. 同步式范型。依赖于问题的自然划分，组织团队成员各自解决问题的一部分，他们之间没有什么主动的交流。

从历史的角度看，最早的软件团队组织是封闭式范型结构，最初称为主程序员团队。这种结构首先由 Harlan Mills 提出，并由 Baker[Bak72] 描述出来。Constantine[Con93]提出的随机式范型是主程序员团队结构的一个变种，主张建立具有独立创新性的团队，其工作方式可恰当地称为创新的无政府状态。尽管自由的软件工作方式是有吸引力的，但在绩效良好的团队中必须将创新能力作为软件工程组织的中心目标。为了建成一支绩效良好的团队，团队成员必须相互信任，团队成员的技能分布必须适合于要解决的问题，并且如果要保持团队的凝聚力，必须将坚持个人己见的人员排除于团队之外。

无论是什么类型的团队，每个项目经理的目标都是帮助建立一支有凝聚力的团队。DeMarco 和 Lister[DeM98]在其论著《Peopleware》中讨论了这个问题：

在商业界，我们往往随便使用团队这个词。任何被分配在一起工作的一组人都可以称为“团队”。但很多这样的小组看起来并不像团队，它们没有统一的对成功的定义，没有任何鲜明的团队精神。它们所缺少的是

一种很珍贵的东西，我们称之为“凝聚力”。

一个有凝聚力的团队是一组团结紧密的人，他们的整体力量大于个体力量的总和……一旦团队开始具有凝聚力，成功的可能性就大大提高。这个团队可以变得不可阻挡，成为成功的象征……他们不需要按照传统的方式进行管理，也不需要去激励。他们已经有了动力。

DeMarco 和 Lister 认为，同一般的团队相比，有凝聚力的团队成员具有更高的生产率和更大的动力。他们拥有统一的目标和共同的文化，而且在很多情况下，“精英意识”使得他们独一无二。

提问 选择软件团队的结构时，应该考虑什么因素？

提问 确定软件团队的结构时，我们有哪些选择？

引述 如果你要做得较好，那就竞争。如果你要做得极好，那就合作。

作者不详

提问 什么是“有凝聚力”的团队？

但是,并非所有的团队都具有凝聚力。事实上,很多团队都受害于 Jackman[Jac98]称之为“团队毒性”的东西。她定义了5个“培育潜在含毒团队环境”的因素:(1)狂乱的工作氛围;(2)引起团队成员间产生摩擦的重大挫折;(3)“碎片式的或协调很差”的软件过程;(4)在软件团队中没有清晰的角色定义;(5)“接连不断地重蹈覆辙”。

提问 为什么团队没有凝聚力?

为了避免狂乱的工作环境,项目经理应该确保团队可以获取完成工作所需的所有信息;而且,主要目标一旦确定下来,除非绝对必要,否则不应该修改。给予软件团队尽可能多的决策权,这样能使团队避免挫败。通过理解将要开发的产品和完成工作的人员,以及允许团队选择过程模型,可以避免选择不适当的软件过程(如不必要的或繁重的工作任务,或没有很好地选择工作产品)。团队本身应该建立自己的责任机制(技术评审是实现此目标的极好方式),并规定一系列当团队成员未能完成任务时的纠正方法。最后,避免失败的关键是建立基于团队的信息反馈方法和解决问题的技术。

引述 只有做或不做,没有尝试。
Yoda
(《星球大战》)

除了 Jackman 描述的5个毒素以外,团队成员的个性不同也给软件团队带来了一系列的问题。有些团队成员性格外向,有些团队成员性格内向。有些人依靠直觉收集信息,从分离的事实中提炼主要概念;有些人则是线性地处理信息,从提供的数据中收集和组织微小的细节。有些团队成员只有在给出逻辑的、有序的论据时,才能做出决策;有些团队成员则依靠直觉,喜欢根据“感觉”做出决策。有些人希望有详细的任务进度计划,使得他们能够按部就班地完成项目的各个部分;有些人则喜欢更自发的环境,在这种环境中,允许开放地争论问题。有些人工作刻苦,在最后期限前很长时间就完成了任务,从而避免了时间逼近所带来的压力;而有些人则经常为了时限的逼近而在最后一分钟加班冲刺。熟练的团队负责人一般都掌握了如何帮助不同个性的人协同工作的方法。如何对待个性不同的人心理学问题,这超出了本书研究的范围^①。不过,重要的是要注意到,识别人员差异是建立有凝聚力的团队的第一步。

22.2.4 敏捷团队

很多软件组织倡导将敏捷软件开发(第5章)作为解决软件项目工作中诸多困扰的一剂良方。回顾一下,敏捷方法学倡导的是:通过尽早地逐步交付软件来使客户满意;组织小型的充满活力的项目团队;采用非正式的方法;交付最小的软件工程工作产品;以及总体开发简易性。

小型的充满活力的项目团队,也称为敏捷团队,这种团队采纳了很多成功的软件项目团队的特性(在上一节内容中谈到的),避免了很多产生问题的毒素。同时,敏捷方法学强调团队成员的个人能力与团队协作精神相结合,这是团队成功的关键因素。对此, Cockburn 和 Highsmith [Coc01a] 这样写道:

如果项目成员足够优秀,那么他们几乎可以采用任何一种过程来完成任务。如果项目成员不够优秀,那么没有任何一种过程可以弥补这个不足。“人员胜过过程”阐明的正是这样的含义。然而,如果缺乏用户和主管人员的支持,也可以毁掉一个项目,即“政策胜过人员”。缺乏支持可以阻止最好的人员完成任务。

在软件项目中,为了充分发挥每个团队成员的能力,并培养有效的合作,敏捷团队是自

^① 关于这些问题(当它们和软件项目团队相关时)的一个很好的介绍可在 [Fer98] 中找到。

组织的。自组织团队不必保持单一的团队结构，而是采用 22.2.3 节讨论的由 Constantine 提出的随机、开放、同步式的范型。

很多敏捷过程模型（如 Scrum）给予敏捷团队相当大的自主权来进行项目管理，可以因工作需要做出技术决定。将计划制定工作压缩到最低程度，并且允许团队自己选择适用的手段（例如，过程、方法和工具），只受业务需求和组织标准的限制。在项目进展过程中，自组织团队关注的是在特定的时间点使项目获益最大的个人能力。为了做到这一点，敏捷团队召开日常团队例会，对当天必须完成的工作进行协调和同步控制。

基于在团队例会中获取的信息，团队能使他们所采用的手段不断适应持续增加的工作。当每一天过去的时候，连续的自组织和协作使团队朝着软件逐步接近的完工的目标前进。

22.2.5 协调和沟通问题

使软件项目陷入困境的原因很多。许多开发项目规模很大，导致复杂性高、混乱、难以协调团队成员间的关系。不确定性是经常存在的，它会引起困扰项目团队的一连串的变更。互操作性已经成为许多系统的关键特性。新的软件必须与已有的软件通信，并遵从系统或产品所施加的预定义约束。

现代软件的这些特征（规模、不确定性和互操作性）确实都存在。为了有效地处理这些问题，必须建立切实可行的方法来协调工作人员之间的关系。为了做到这一点，需要建立团队成员之间以及多个团队之间的正式的和非正式的交流机制。正式的交流机制是通过“文字、各级会议及其他相对而言非交互的和非个人的交流渠道”[Kra95]来实现的。非正式的交流机制则更加个人化。软件团队的成员在遇到特殊情况时交流意见，出现问题时请求帮助，而且在日常工作中彼此之间互相影响。

关键点 敏捷团队是自组织的团队，拥有制定计划和做技术决定的自主权。

引述 集体所有权只不过是如下观念的产物：产品属于（敏捷）团队，而不属于团队中的个人。

Jim Highsmith

SafeHome 团队结构

[场景] SafeHome 软件项目启动之前，Doug Miller 的办公室。

[人物] Doug Miller，SafeHome 软件工程团队经理；Vinod Raman，Jamie Lazar 及其他产品软件工程团队成员。

[对话]

Doug：你们看过市场销售部准备的有关 SafeHome 的基本信息了吗？

Vinod（一边看着同事，一边点头）：是的，但我们有很多问题。

Doug：过一会儿再讨论这些问题。我们先来讨论一下应该如何组织一个团队，哪些人应该负责……

Jamie：Doug，我对敏捷方法非常感兴趣，

我想我们应该是一个自组织的团队。

Vinod：我同意。给定一个我们都能胜任的严格的期限和某些不确定性（笑），看起来是一种正确的方式。

Doug：我赞同，但是你们知道如何操作吗？

Jamie（边笑边说，好像在背诵什么）：我们做出战术决定，确定由谁做、做什么、什么时间做。但按时交付产品是我们的责任。

Vinod：还有质量。

Doug：很正确，但是记住还有约束。市场部决定要生产的软件的增量，当然这要征求我们的意见。

Jamie: 还有?

Doug: 还有, 要使用 UML 作为我们的建模方法。

Vinod: 但要保持无关的文档减到最少。

Doug: 那谁和我联络?

Jamie: 我们确定 Vinod 作为技术负责人, 因为他的经验最丰富。因此, Vinod 是你的联络人, 但你应该自由地与每个人交流。

Doug: 别担心, 我会的。

22.3 产品

从软件项目一开始, 软件项目经理就面临着进退两难的局面。需要定量地估算成本和有组织地计划项目的进展, 但却没有可靠的信息可以使用。虽然对软件需求的详细分析可以提供估算所需的信息, 但需求分析常常需要数周甚至数月的时间才能完成。更糟糕的是, 需求可能是不固定的, 随着项目的进展经常会发生变化。然而, 计划总是“眼前”就需要的!

不管喜欢与否, 从项目一开始, 就要研究应该开发哪些产品以及要解决哪些问题。至少, 我们要建立和界定产品的范围。

22.3.1 软件范围

软件项目管理的第一项活动是确定软件范围。软件范围是通过回答下列问题来定义的:

项目环境。要开发的软件如何适应于大型的系统、产品或业务环境, 该环境下要施加什么约束?

信息目标。软件要产生哪些客户可见的数据对象作为输出? 需要什么数据对象作为输入?

功能和性能。软件要执行什么功能才能将输入数据转换成输出数据? 软件需要满足什么特殊的性能要求?

软件项目范围在管理层和技术层都必须是无歧义的和可理解的。对软件范围的描述必须是界定的。也就是说, 要明确给出定量的数据 (例如, 并发用户数、邮件列表的长度、允许的最大响应时间); 说明约束和限制 (例如, 产品的成本要求会限制内存的大小), 并描述其他的调节因素 (例如, 期望的算法能被很好地理解, 并采用 Java 实现)。

建议 如果你不能把握待开发软件的某个特征, 就将该特征作为一个项目风险列出 (第 26 章)。

22.3.2 问题分解

问题分解, 有时称为问题划分或问题细化, 它是软件需求分析 (第 7 ~ 10 章) 的核心活动。在确定软件范围的活动中, 并不试图去完全分解问题, 只是分解其中的两个主要方面: (1) 必须交付的功能和内容 (信息); (2) 所使用的过程。

在面对复杂的问题时, 人们常常采用分而治之的策略。简单地说, 就是将一个复杂的问题划分成若干更易处理的小问题。这是项目计划开始时所采用的策略。在开始估算 (第 24 章) 前, 必须对软件范围中所描述的软件功能进行评估和细化, 以提供更多的细节。因为成本和进度估算都是面向功能的, 所以对功能进行某种程度的分解是很有益的。同样, 为了对软件生成的信息提供合理的解释, 要将主要的内容对象或数据对象分解为各自的组成部分。

建议 为了制定合理的项目计划, 你必须对问题进行分解。问题分解可以使用一系列的功能或者用例来进行, 对于敏捷开发来说, 使用的是用户故事。

性、突破性的新技术、难以相处的客户、明显的复用潜力等),可能就要选择其他过程模型^①。

一旦选定了过程模型,就要根据所选的过程模型对过程框架做适应性修改。但在所有情况下,前面讨论过的通用框架活动都可以使用。它既适用于线性模型,也适用于迭代和增量模型、演化模型,甚至是并发模型或构件组装模型。过程框架是不变的,是软件组织进行所有工作的基础。

但实际的工作任务是不同的。当项目经理问:“我们如何完成这个框架活动”时,就意味着过程分解开始了。例如,一个小型的比较简单的项目在沟通活动中可能需要完成下列工作任务:

1. 列出需澄清的问题清单。
2. 与利益相关者会面商讨需澄清的问题。
3. 共同给出范围陈述。
4. 和所有相关人员一起评审范围陈述。
5. 根据需要修改范围陈述。

这些事件可能在不到48小时的时间内发生。这是一种过程分解方式,这种方式适用于小型的比较简单的项目。

现在,考虑一个更复杂的项目,它的范围更广,具有更重要的商业影响。这样一个项目在沟通中可能需要完成下列工作任务:

1. 评审客户需求。
2. 计划并安排与全体利益相关者召开正式的、有人主持的会议。
3. 研究如何说明推荐的解决方案和现有的方法。
4. 为正式会议准备一份“工作文档”和议程。
5. 召开会议。
6. 共同制定能够反映软件的数据、功能和行为特性的微型规格说明。或者,从用户的角度出发建立描述软件的用例。
7. 评审每一份微型规格说明或用例,确认其正确性、一致性和无歧义性。
8. 将这些微型规格说明组装起来形成一份范围文档。
9. 和所有相关人员一起评审范围文档或用例集。
10. 根据需要修改范围文档或用例。

两个项目都执行了我们称之为沟通的框架活动,但第一个项目团队的软件工作任务只是第二个项目团队的一半。

22.5 项目

为了成功地管理软件项目,我们必须了解可能会出现什么毛病,以便能避免这些问题。在一篇关于软件项目的优秀论文中,John Reel[REE99]定义了若干预示信息系统项目正处于危险状态的信号。有些时候,软件人员不理解客户的需要,这导致产品范围定义得很糟糕。在一些项目中,变更没有得到很好的管理。有时,所选的技术发生了变化,或者业务需求改变了,或者失去了

关键点 过程框架建立了项目计划的纲要,并通过分配适合项目的一系列任务对其进行调整。

提问 什么信号表示软件项目正处于危险状态?

^① 回忆一下,项目特性对项目团队的结构也有相当大的影响,见22.2.3节。

赞助。管理者可能设定了不切实际的最后期限，或者最终用户抵制这个新系统。有些情况下，项目团队不具有所需的技能。最后还可能是，有些开发人员似乎从来没有从自己的错误中学习。

在讨论特别困难的软件项目时，疲惫不堪的从业人员常常提及“90-90 规则”：系统前面 90% 的任务会花费所分配总工作量和时间的 90%，系统最后 10% 的任务也会花费所分配总工作量和时间的 90% [Zah94]。导致该 90-90 规则的根源就在上面列出的“信号”中。

这太消极了！管理者如何避免上面提到的这些问题呢？Reel [Ree99] 针对软件项目提出了以下易于理解的方法，共包含 5 部分。

1. 在正确的基础上开始工作。通过以下两点来实现：首先努力（非常努力）地正确理解要解决的问题，然后为每个参与项目的人员设置现实的目标和期望。这一点又通过组建合适的开发团队（22.2.3 节）并给予团队工作时所需的自由、权力和技术而得到加强。
2. 保持动力。很多项目的启动都有一个良好的开端，但是，后来慢慢地开始瓦解。为了维持动力，项目经理必须采取激励措施使人员变动量保持绝对最小，团队应该重视它完成的每项任务的质量，而高层管理应该尽可能不干涉团队的工作。^①
3. 跟踪进展。对于软件项目而言，当工作产品（如模型、源代码、测试用例集）正在产生或被认可（通过技术评审）时，跟踪项目进展要作为质量保证活动的一部分。此外，可以收集软件过程和项目测量（第 23 章）数据，然后对照软件开发组织的平均数据来评估项目的进展。
4. 做出英明的决策。总体上，项目经理和软件团队的决策应该“保持项目的简单性”。只要有可能，就使用商用成品软件或现有的软件构件或模式，可以采用标准方法时避免定制接口，识别并避免显而易见的风险，以及分配比你认为的时间更多的时间来完成复杂或有风险的任务（你需要每一分钟）。
5. 进行事后分析。建立统一的机制，从每个项目中获取可学习的经验。评估计划的进度和实际的进度，收集和分析软件项目度量数据，从团队成员和客户处获取反馈，并记录所有的发现。

22.6 W⁵HH 原则

Barry Boehm [Boe96] 在其关于软件过程和项目的优秀论文中指出：“你需要一个组织原则，对它进行缩减来为简单的项目提供简单的（项目）计划。”Boehm 给出了一种方法，该方法描述项目的目标、里程碑、进度、责任、管理和技术方法以及需要的资源。他称之为 W⁵HH 原则。这种方法通过提出一系列问题，来导出对关键项目特性以及项目计划的定义：

为什么（Why）要开发这个系统？所有利益相关者都应该了解软件工作的商业理由是否有效。该系统的商业目的值得花费这些人、时间和金钱吗？

将要做什么（What）？定义项目所需的任务集。

什么时候（When）做？团队制定项目进度，标识出何时开展项目任务以及何时到达里程碑。

提问 我们如何定义关键的项目特性？

^① 这句话的意思是：将官僚主义减少到最低程度，取消无关的会议，不再强调教条地依附于过程和项目规则。团队应该是自组织的，拥有自治权。

某功能由谁(Who)负责?规定软件团队每个成员的角色和责任。

他们的机构组织位于何处(Where)?并非所有角色和责任均属于软件团队,客户、用户和其他利益相关者也有责任。

如何(How)完成技术工作和管理工作?一旦确定了产品范围,就必须定义项目的管理策略和技术策略。

每种资源需要多少(How much)?对这个问题,需要在对前面问题回答的基础上通过估算(第24章)而得到。

Boehm的W³HH原则可适用于任何规模和复杂度的软件项目。给出的问题为你和你的团队提供了很好的计划大纲。

22.7 关键实践

Airlie Council^①提出了一组“基于性能管理的关键软件实践”。这些实践一直“被高度成功的软件项目和组织(它们的‘底线’性能大大优于产业界的平均水平)普遍采用,并被认为的确是关键的”[Air99]。

关键实践^②包括:基于度量的项目管理(第23章),成本及进度的经验估算(第24和25章),获得价值跟踪(第25章),根据质量目标跟踪缺陷(第15和16章),人员计划管理(第22.2节)。每一项关键实践都贯穿于本书的第四部分。

软件工具 | 项目管理的软件工具

这里列出的都是通用工具,适用于大部分项目管理活动。而特定的项目管理工具,如计划工具、评估工具和风险分析工具,将在后续章节中讲述。

[代表性工具]^③

- Projectmanagement.com (<http://www.projectmanagement.com>) 已经为项目经理开发了一套实用的检查表。

projectmanagement.com) 已经为项目经理开发了一套实用的检查表。

- Ittoolkit.com(www.ittoolkit.com) “收集了很多计划指南、过程模板和精巧的工作表”, 可以从光盘上获取。

习题与思考题

- 22.1 基于本章给出的信息和自己的经验,列举出能够增强软件工程师能力的“10条戒律”。即列出10条指导原则,使得软件人员能够在工作中发挥其全部潜力。
- 22.2 SEI的人员能力成熟度模型(People-CMM)定义了培养优秀软件人员的“关键实践域”(KPA)。你的老师将为你指派一个关键实践域,请你对它进行分析和总结。
- 22.3 描述3种现实生活中的实际情况,其中客户和最终用户是相同的人。也描述3种他们是不同人的情况。
- 22.4 高级管理者所做的决策会对软件工程团队的效率产生重大影响。请列举5个实例来说明这

① Airlie Council 由美国国防部组织的软件工程专家组组成,其目的是开发在软件项目管理和软件工程中最佳的实践指南。关于最佳实践的更多信息参见 http://www.swqual.com/e_newsletter.html。

② 这里只讨论与“项目完整性”有关的关键实践。

③ 这里提到的工具只是此类工具的例子,并不代表本书支持采用这些工具。在大多数情况下,工具名称被各自的开发者注册为商标。

一点。

- 22.5 温习 Weinberg 的书 [Wei86], 并写出一份 2 ~ 3 页的总结, 说明在使用 MOI 模型时应该考虑的问题。
- 22.6 在一个信息系统组织中, 你被指派为项目经理。你的工作是开发一个应用程序, 该程序类似于你的团队已经做过的项目, 只是规模更大而且更复杂。需求已经由用户写成文档。你会选择哪种团队结构? 为什么? 你会选择哪(些)种软件过程模型? 为什么?
- 22.7 你被指派为一个小型软件产品公司的项目经理。你的工作是开发一个有突破性的产品, 该产品结合了虚拟现实的硬件和高超的软件。家庭娱乐市场的竞争非常激烈, 因而完成这项工作的压力很大。你会选择哪种团队结构? 为什么? 你会选择哪些软件过程模型? 为什么?
- 22.8 你被指派为一个大型软件产品公司的项目经理。你的工作是管理该公司已被广泛使用的字处理软件的新版本的开发。由于竞争激烈, 因此已经规定了紧迫的最后期限, 并对外公布。你会选择哪种团队结构? 为什么? 你会选择哪些软件过程模型? 为什么?
- 22.9 在一个为遗传工程领域服务的公司中, 你被指派为软件项目经理。你的工作是管理一个软件新产品的开发, 该产品能够加速基因分类的速度。这项工作面向研究及开发的, 但其目标是在下一年度内生产出产品。你会选择哪种团队结构? 为什么? 你会选择哪些软件过程模型? 为什么?
- 22.10 要求开发一个小型应用软件, 它的作用是分析一所大学开设的每一门课程, 并输出课程的平均成绩(针对某个学期)。写出该问题的范围陈述。
- 22.11 给出 22.3.2 节中讨论的页面布局功能的第一级功能分解。

扩展阅读与信息资源

项目管理研究所 (PMI) 出版的书 (《 Guide to the Project Management Body of Knowledge 》, 4th ed., PMI, 2009) 介绍了项目管理的所有重要方面。Wysocki (《 Effective Software Project Management: Traditional, Agile, Extreme 》, 6th ed., Wiley, 2011)、Slinger 和 Broderick (《 The Software Project Manager's Bridge to Agility 》, Addison-Wesley, 2008)、Bechtold (《 Essentials of Software Project Management 》, 2nd ed., Management Concepts, 2007)、Stellman 和 Greene (《 Applied Software Project Management 》, O'Reilly, 2005) 以及 Berkun (《 Making Things Happen: Mastering Project Management-Theory in Practice 》, O'Reilly, 2008) 讲授了基本技能, 并为所有的软件项目管理任务提供了详细的指南。McConnell (《 Professional Software Development 》, Addison-Wesley, 2004) 提供了实用的建议, 指导如何获得“更短的进度计划、更高质量的产品和更成功的项目”。Henry (《 Software Project Management 》, Addison-Wesley, 2003) 为所有项目经理提供了现实的建议。

Tom DeMarco 和他的同事 (《 Adrenaline Junkies and Template Zombies 》, Dorset House, 2008) 对每种软件项目中会遇到的人员模式都给出了有深入见解的处理方法。由 Weinberg 所著的一套 4 册的系列丛书 (《 Quality Software Management 》, Dorset House, 1992, 1993, 1994, 1996) 介绍了基本的系统思想和管理概念, 讲解了如何有效地使用测量, 并且说明了“一致的行为”, 即建立管理者需要、技术人员需要以及商业需要之间的协调关系。它不仅为新的管理者同时也为有经验的管理者提供了有用的信息。Futrell 和他的同事 (《 Quality Software Project Management 》, Prentice-Hall, 2002) 提出了大量的项目管理处理方法。Neill 和他的同事 (《 Antipatterns: Managing Software Organizations 》, 2nd ed., Auerbach Publications, 2011) 以及 Brown 和他的同事 (《 Antipatterns in Project Management 》, Wiley, 2000) 在其所著的书中讨论了软件项目管理期间不能做的事情。

Brooks (《 The Mythical Man-Month 》, Anniversary Edition, Addison-Wesley, 1995) 更新了他的杰作, 给出了关于软件项目及管理问题的新见解。McConnell (《 Software Project Survival Guide 》, Microsoft Press, 1997) 为那些必须管理软件项目的人提供了卓越的有实效的行动指南。Purba 和 Shah

(《How to Manage a Successful Software Project》, 2nd ed., Wiley, 2000) 提供了很多案例研究, 指出为什么一些项目能成功, 而另外一些项目却会失败。Kerzner(《Project Management: A Systems Approach to Planning Scheduling and Controlling》, 10th ed., Wiley, 2009) 和 Bennatan(《On Time Within Budget》, 3rd ed., Wiley, 2000) 在其所著的书中为软件项目经理提供了实用的提示和指导。Weigers(《Practical Project Initiation》, Microsoft Press, 2007) 为软件项目的成功实施提供了实用的指南。

可以证明, 软件项目管理中最重要的方面是人员管理。Cockburn(《Agile Software Development》, Addison-Wesley, 2002) 给出了关于软件人员的论述, 这是到目前为止关于软件人员的最好的论述之一。DeMarco 和 Lister[DeM98] 撰写了关于软件人员和软件项目的最权威著作。此外, 关于这一主题, 近年来出版的如下书籍值得一读:

Cantor, M., 《Software Leadership: A Guide to Successful Software Development》, Addison-Wesley, 2001。

Carmel, E., 《Global Software Teams: Collaborating Across Borders and Time Zones》, Prentice Hall, 1999。

Chandler, H. M., 《Game Production Handbook》, 2nd ed., Charles River Media, 2008。

Constantine, L., 《Peopleware Papers: Notes on the Human Side of Software》, Prentice Hall, 2001。

Ebert, C., 《Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing》, Wiley-IEEE Computer Society, 2011。

Fairley, R. E., 《Managing and Leading Software Projects》, Wiley-IEEE Computer Society, 2009。

Garton, C., and Wegryn, K., 《Managing Without Walls》, McPress, 2006。

Humphrey, W. S., and Over, J. W., 《Leadership, Teamwork, and Trust: Building a Competitive Software Capability》, Addison-Wesley, 2011。

Humphrey, W. S., 《Managing Technical People: Innovation, Teamwork, and the Software Process》, Addison-Wesley, 1997。

Humphrey, W. S., 《TSP-Coaching Development Teams》, Addison-Wesley, 2006。

Jones, P. H., 《Handbook of Team Design: A Practitioner's Guide to Team Systems Development》, McGraw-Hill, 1997。

Karolak, D. S., 《Global Software Development: Managing Virtual Teams and Environments》, IEEE Computer Society, 1998。

Misrik, I., et al., 《Collaborative Software Engineering》, Springer, 2010。

Peters, L., 《Getting Results from Software Development Teams》, Microsoft Press, 2008。

Whitehead, R., 《Leading a Software Development Team》, Addison-Wesley, 2001。

以下列出的一些畅销的“管理”书籍并不和软件世界特别相关, 有的还过于简单、过于概括: Kanter(《Confidence》, Three Rivers Press, 2006), Covey(《The 8th Habit》, Free Press, 2004), Bossidy(《Execution: The Discipline of Getting Things Done》, Crown Publishing, 2002), Drucker(《Management Challenges for the 21st century》, Harper Business, 1999), Buckingham 和 Coffman(《First, Break All the Rules: What the World's Greatest Managers Do Differently》, Simon and Schuster, 1999), 以及 Christensen(《The Innovator's Dilemma》, Harvard Business School Press, 1997)。这些书强调由快速变化的经济定义的“新规则”。比较老的书如《Who Moved My Cheese?》《The One-Minute Manager》和《In Search of Excellence》, 它们仍然很有价值, 能够帮助你更有效地管理人员和项目。

在网上可以获得大量的关于软件项目管理的信息。最新的参考文献可在 SEPA 网站 www.mhhe.com/pressman 下的“software engineering resources”中找到。

过程度量与项目度量

要点浏览

概念：软件过程度量和项目度量是定量的测量，这些测量能使你更深入地了解软件过程的功效，以及使用该过程作为框架进行开发的项目的功效。做度量时，首先要收集基本的质量数据和生产率数据，然后分析这些数据、与过去的平均值进行比较，通过评估来确定是否有质量的改进和生产率的提高。度量也可以用来查明问题区域，确定恰当的补救措施，从而改进软件过程。

人员：软件度量由软件管理者来分析和评估。测量数据通常由软件工程师来收集。

重要性：如果不进行测量，就只能根据主观评价来做判断。通过测量，可以发现趋势（好的或坏的），可以更好地进行估

算，并且随着时间的推移，软件能够获得真正的改进。

步骤：首先确定一组数量有限的易于收集的过程测量、项目测量和产品测量。通常使用面向规模或面向功能的度量对这些测量进行规范化。然后对测量结果进行分析，与该组织以前完成的类似项目的平均数据进行比较。最后评估趋势，给出结论。

工作产品：一组软件度量，它们提供了对过程深入透彻的认识和对项目的理解。

质量保证措施：通过采用一致而简单的测量计划来保障，但该计划绝对不能用于对个人表现的评估、奖励或惩罚。

通过提供目标评估的机制，测量能使我们对过程和项目有更深入的了解。Lord Kelvin 曾经说过：

当你能够测量你所说的事物，并能用数字表达时，你就对它有了一定的了解；如果不能测量它，也不能用数字表达时，就说明你对它的了解还很贫乏，不能令人满意。后者可能是知识的起点，但你在思想上还远远没有达到科学的境地。

软件工程界已经认可了 Lord Kelvin 的话。但这并不是一帆风顺，也不是只有一点点争论。

将测量应用于软件过程，目的是持续改进软件过程。也可以将测量应用于整个软件项目，辅助进行估算、质量控制、生产率评估及项目控制。软件工程师可以使用测量来帮助评估工作产品的质量，在项目进展过程中辅助进行战术决策。

从软件过程以及使用该过程进行开发的项目出发，软件团队主要关注生产率度量和质量度量——前者是对软件开发“输出”的测量，它是投入的工作量和时间的函数，后者是对所生产的工作产品“适用性”的测量。为了进行计划和估算，需要了解历史数据。以往项目的

软件开发生产率是多少？开发出来的软件质量怎么样？怎样利用以往的生产率数据和质量数据推断现在的生产率和质量？这些数据如何帮助我们更精确地计划和估算？

在 Park、Goethert 和 Florac[Par96b] 的关于软件测量的指导手册中，他们讨论了进行测量的理由：（1）通过刻画而“获得对过程、产品、资源和环境的了解，建立同未来评估进行比较的基线”；（2）通过评价来确定“相对于计划的状况”；（3）“通过理解过程和产品间的关系，并构建这些关系的模型来进行预测”；（4）“通过识别难点、根本原因、低效率和其他提高产品质量和过程性能的机会来进行改进”。

测量是一个管理工具，如果能正确地使用，它将为项目管理者提供洞察力。因此，测量能够帮助项目管理者 and 软件团队制定出使项目成功的决策。

23.1 过程领域和项目领域中的度量

过程度量的收集涉及所有的项目，要经历相当长的时间，目的是提供能够引导长期的软件过程改进的一组过程指标。项目度量使得软件项目管理者能够：（1）评估正在进行中的项目的状态；（2）跟踪潜在的风险；（3）在问题造成不良影响之前发现它们；（4）调整工作流程或任务；（5）评估项目团队控制软件工作产品质量的能力。

测量数据由项目团队收集，然后被转换成度量数据在项目期间使用。测量数据也可以传给那些负责软件过程改进的人员。因此，很多相同的度量既可用于过程领域，又可用于项目领域。

23.1.1 过程度量和软件过程改进

改进任何过程的唯一合理方法就是测量该过程的特定属性，再根据这些属性建立一组有意义的度量，然后使用这组度量提供的指标来导出过程改进策略。但是在讨论软件度量及其对软件过程改进的影响之前，必须注意到：过程仅是众多“改进软件质量和组织性能的控制因素”中的一种 [Pau94]。

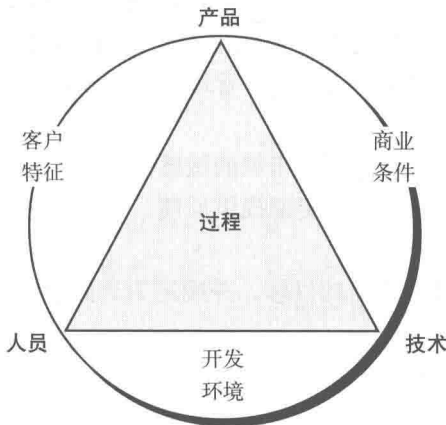


图 23-1 软件质量和组织有效性的决定因素 [Pau94]

关键概念	
缺陷排除效率	
(DRE)	
功能点 (FP)	
测量	
度量	
争论	
基线	
制定大纲	
面向功能	
基于 LOC 的	
度量	
面向对象	
私有和公有	
过程	
生产率	
项目	
面向规模	
软件质量	
面向用例	

在图 23-1 中,过程位于三角形的中央,连接了三个对软件质量和组织绩效有重大影响的因素。其中,人员的技能和动力 [Boe81] 被认为是对质量和绩效影响最大的因素,产品复杂性对质量和团队绩效也有相当大的影响,过程中采用的技术(即软件工程方法和工具)也有一定的影响。

另外,过程三角形位于环境条件圆圈内,环境条件包括:开发环境(如集成的软件工具)、商业条件(如交付期限、业务规则)、客户特征(如交流和协作的难易程度)。

软件过程的功效只能间接地测量。也就是说,根据从过程中获得的结果来导出一组度量。这些结果包括:在软件发布之前发现的错误数的测度,提交给最终用户并由最终用户报告的缺陷的测度,交付的工作产品(生产率)的测度,花费的工作量的测度,花费时间的测度,与进度计划是否一致的测度,以及其他测度。也可以通过测量特定软件工程任务的特性来导出过程度量。例如,测量第 2 章中所描述的普适性活动和一般软件工程活动所花费的工作量和时间。

Grady [Gra92] 认为不同类型过程数据的使用可以分为“私有的和公有的”。软件工程师可能对他个人收集的度量的使用比较敏感,这是很自然的事。这些数据对本人应该是私有的,是仅供本人参考的指标。私有度量的例子有:缺陷率(个人)、缺陷率(软件构件)和开发过程中发现的错误数。

“私有过程数据”的观点与 Humphrey [Hum05] 所提出的个人软件过程方法(第 4 章)相一致。Humphrey 认为软件过程改进能够也应该开始于个人级。私有过程数据是改进自身软件工程方法的重要驱动力。

有些过程度量是软件项目团队私有的,但对团队所有成员是公有的。例如,主要软件功能(由多个开发人员共同完成)的缺陷报告、技术评审发现的错误,以及每个构件或功能的代码行数或功能点数^①。团队评审这些数据,以找出能够提高团队效能的指标。

公有度量一般吸收了原本属于个人或团队的私有信息。收集和评估项目级的缺陷率(绝对不能归咎于某个人)、工作量、时间以及相关的数据,来找出能够改善组织的过程性能的指标。

软件过程度量对于组织提高其过程成熟度的整体水平能够提供很大的帮助。不过,与其他所有度量一样,软件过程度量也可能被误用,产生的问题比它们所能解决的问题更多。Grady [Gra92] 提出了一组“软件度量规则”。管理者和开发者在制定过程度量大纲时,这些规则都适用:

- 解释度量数据时使用常识,并考虑组织的敏感性。
- 向收集测量和度量的个人及团队定期提供反馈。
- 不要使用度量去评价个人。
- 与开发者和团队一起设定清晰的目标,并确定为达到这些目标需要使用的度量。
- 不要用度量去威胁个人或团队。

关键点 软件人员的技能和动力是影响软件质量的最重要因素。

引述 软件度量让你知道什么时候笑,什么时候哭。

Tom Gilb

提问 对软件度量的私有使用和公有使用有什么不同?

提问 当我们收集软件度量时,应该采用什么指导原则?

① 代码行和功能点度量将在第 23.2.1 和 23.2.2 节中讨论。

- 指出问题区域的度量数据不应该被“消极地”看待，这些数据仅仅是过程改进的指标。
- 不要在某一个别的度量上纠缠，而无暇顾及其他重要的度量。

随着一个组织更加得心应手地收集和使用过程度量，简单的指标获取方式会逐渐被更精确的称为统计软件过程改进（statistical software process improvement, SSPI）的方法所取代。本质上，SSPI 使用软件失效分析方法来收集在应用软件、系统或产品的开发及使用过程中所遇到的所有错误及缺陷信息^①。

23.1.2 项目度量

软件过程度量用于战略目的，而软件项目测量则用于战术目的。也就是说，项目管理者 and 软件项目团队通过使用项目度量及从中导出的指标，可以改进项目工作流程和技术活动。

在大多数软件项目中，项目度量的第一次应用是在估算阶段。那些从以往项目中收集的度量可以作为当前软件工作的工作量及时间估算的基础。随着项目的进展，将所花费的工作量及时间的测量与最初的估算值（及项目进度）进行比较。项目管理者可以使用这些数据来监控项目的进展。

随着技术工作的启动，其他项目度量也开始有意义了。生产率可以根据创建的模型、评审时间、功能点以及交付的源代码行数来测量。此外，对每个软件工程任务中发现的错误也要进行跟踪。在软件从需求到设计的演化过程中，需要收集技术度量来评估设计质量，并提供若干指标，这些指标将会影响代码生成及测试所采用的方法。

项目度量的目的是双重的。首先，利用度量能够对开发进度进行必要的调整，以避免延迟，并减少潜在的问题和风险，从而使开发时间减到最短。其次，项目度量可用于在项目进行过程中评估产品质量，必要时可调整技术方法以提高质量。

提问 在项目中，我们应该如何使用度量？

随着质量的提高，缺陷会越来越少。随着缺陷数的减少，项目所需的修改工作量也会减少，从而降低项目的总体成本。

SafeHome 建立度量方法

[场景] SafeHome 软件项目即将启动，在 Doug Miller 的办公室。

[人物] Doug Miller, SafeHome 软件工程项目团队经理；Vinod Raman 和 Jamie Lazar, 产品软件工程团队成员。

[对话]

Doug: 在项目工作开始之前，我想你们应该定义并收集一组简单的度量。首先，必须确定目标。

Vinod (皱着眉头): 以前，我们从来没有

做过这些，并且……

Jamie (打断他的话): 基于时间线的管理已经讨论过了，我们根本没有时间。度量到底有什么好处？

Doug (举手示意停止发言): 大家且慢，停一下。正因为我们以前从来没有做过度量，所以现在更要开始做。并且我说的度量工作根本不会占用很多时间，事实上，它只会节省我们的时间。

Vinod: 为什么？

^① 在本书中，错误（error）是指软件工程项目中的瑕疵（flaw），这些瑕疵在交付给最终用户之前已经被发现。而缺陷（defect）是指交付给最终用户之后才发现的瑕疵。应该注意到，其他人并没有进行这样的区分。

Doug：你看，随着我们的产品更加智能化，变得支持 Web、移动端等，我们将要做更多的内部软件工程工作。我们需要了解软件开发的过程，并改进过程，使我们能够更好地开发软件。要实现这一点，唯一的方法就是测量。

Jamie：但是我们的时间很紧迫，Doug。我不赞同太多琐碎的文字工作，我们需要时间来完成工作，而不是收集数据。

Doug：Jamie，工程师的工作包括收集数据、评估数据、使用评估结果来改进产品和过程。我错了吗？

Jamie：不，但是……

Doug：如果我们限定要收集的测量数不超过 5 个或 6 个，并集中关注质量方面，那

会怎么样？

Vinod：没有人能够反对高质量……

Jamie：对，但是，我不知道，我仍然认为这是不必要的。

Doug：在这个问题上，请听我的！关于软件度量你们了解多少？

Jamie（看着 Vinod）：不多。

Doug：这是一些 Web 参考资料，花几个小时读完。

Jamie（微笑着）：我还认为你说的这件事不会花费任何时间。

Doug：花费在学习上的时间绝对不会浪费，去做吧！然后我们要建立一些目标，提几个问题，定义我们需要收集的度量。

23.2 软件测量

物质世界中的测量可以分为两种方法：直接测量（如螺栓的长度）和间接测量（如螺栓的“质量”，通过统计废品数量来测量）。软件度量也可以这样来划分。

软件过程的直接测量包括花费的成本和工作量。产品的直接测量包括产生的代码行（LOC）、运行速度、存储容量以及某段时间内报告的缺陷；产品的间接测量包括功能、质量、复杂性、效率、可靠性、可维护性，以及许多在第 15 章中谈到的其他“产品特性”。

构造软件所需的成本和工作量、产生的代码行数以及其他直接测量都是相对容易收集的，只要事先建立特定的测量协议即可。但是，软件的质量和功能、效率或可维护性则很难获得，只能间接地测量。

我们已将软件度量范围分为过程度量、项目度量和产品度量。注意，产品度量对个人来讲是私有的，常常将它们合并起来生成项目度量，而项目度量对软件团队来说是公有的。再将项目度量联合起来可以得到整个软件组织公有的过程度量。但是，一个组织如何将来自不同个人或项目的度量结合起来呢？

为了说明这个问题，看一个简单的例子。两个不同项目团队中的人将他们在软件过程中发现的所有错误进行了记录和分类。然后，将这些个人的测量结合起来就产生了团队的测量。在软件发布前，团队 A 在软件过程中发现了 342 个错误，团队 B 发现了 184 个错误。所有其他情况都相同，那么在整个过程中哪个团队能更有效地发现错误呢？由于不了解项目的规模或复杂性，所以不能回答这个问题。不过，如果度量采用规范化的方法，就有可能产生在更大的组织范围内进行比较的软件度量。

引述 并非能够被计算的每件事物都有价值，也并非有价值的每件事物都能够被计算。

Albert Einstein

建议 因为有很多因素会影响软件工作，因此不要用度量去比较个人或团队。

23.2.1 面向规模的度量

面向规模的软件度量是通过对质量和生产率的测量进行规范化后得到的，而这些测量都是根据开发过的软件的规模得到的。如果软件组织一直在做简单记录，就会产生一个如图 23-2 所示的面向规模测量的表。该表列出了在过去几年中完成的每一个软件开发项目及其相关的测量数据。查看 alpha 项目的数据（图 23-2）：花费了 24 人月的工作量，成本为 168000 美元，产生了 12100 行代码。需要提醒大家的是，表中记录的工作量和成本涵盖了所有软件工程活动（分析、设计、编码及测试），而不仅仅是编码。有关 alpha 项目更进一步的信息包括：产生了 365 页文档，在软件发布之前发现了 134 个错误，在软件发布给客户之后运行的第一年中遇到了 29 个缺陷，有 3 个人参加了 alpha 项目的软件开发工作。

项目	代码行	工作量	成本 (千美元)	文档页数	错误	缺陷	人员
alpha	12100	24	168	365	134	29	3
beta	27200	62	440	1224	321	86	5
gamma	20200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

图 23-2 面向规模的度量

为了得到能和其他项目同类度量进行比较的度量，你可以选择代码行作为规范化值。根据表中包含的基本数据，每个项目都能得到一组简单的面向规模的度量：

- 每千行代码（KLOC）的错误数
- 每千行代码（KLOC）的缺陷数
- 每千行代码（KLOC）的成本
- 每千行代码（KLOC）的文档页数

此外，还能计算出其他有意义的度量：

- 每人月错误数
- 每人月千行代码数
- 每页文档的成本

面向规模的度量是否是软件过程测量的最好方法，对此并没有普遍一致的观点。大多数争议都围绕着使用代码行（LOC）作为关键的测量是否合适。LOC 测量的支持者声称：LOC 是所有软件开发项目的“产物”，并且很容易进行计算；许多现有的软件估算模型都是使用 LOC 或 KLOC 作为关键的输入；而且已经有大量的文献和数据都涉及 LOC。另一方面，反对者则认为，LOC 测量依赖于程序设计语言；当考虑生产率时，这种测量对设计得很好但较短的程序会产生不利的评价；它们不适用于非过程语言；而且在估算时需

关键点 面向规模的度量已经得到了广泛的应用，但对其有效性和适用性的争论一直在持续。

要一些可能难以得到的信息（例如，计划人员必须在分析和设计远未完成之前，就要估算出将产生的 LOC）。

23.2.2 面向功能的度量

面向功能的软件度量以功能（由应用程序提供）测量数据作为规范化值。应用最广泛的面向功能的度量是功能点（Function Point，FP）。功能点是根据软件信息域的特性及复杂性来计算的。

与 LOC 测量一样，功能点测量也是有争议的。支持者认为 FP 与程序设计语言无关，对于使用传统语言和非过程语言的应用系统来说，它都是比较理想的，而且它所依据的数据是在项目开发初期就可能得到的数据。因此，FP 是一种更有吸引力的估算方法。反对者则声称这种方法需要某种“熟练手法”，因为计算的依据是主观的而非客观的数据，信息域（及其他方面）的数据可能难以在事后收集。而且，FP 没有直接的物理意义，它仅仅是一个数字而已。

23.2.3 调和代码行度量和功能点度量

代码行和功能点之间的关系依赖于实现软件所采用的程序设计语言及设计的质量。很多研究试图将 FP 测量和 LOC 测量关联起来。表 23-1[⊖] [QSM02] 给出了在不同的程序设计语言中实现一个功能点所需的平均代码行数的粗略估算。

表 23-1 各种程序设计语言实现一个功能点所需的代码行数的粗略估算

程序设计语言	LOC/FP			
	平均值	中值	低值	高值
Ada	154	—	104	205
ASP	56	50	32	106
Assembler	337	315	91	694
C	148	107	22	704
C++	59	53	20	178
C#	58	59	51	704
COBOL	80	78	8	400
ColdFusion	68	56	52	105
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Focus	43	42	32	56
FORTRAN	90	118	35	—
FoxPro	32	35	25	35
HTML	43	42	35	53
Informix	42	31	24	57
J2EE	57	50	50	67
Java	55	53	9	214
JavaScript	54	55	45	63

⊖ 表中列出的数据是 Quantitative Software Management (www.qsm.com) 所开发数据的缩略版，已得到他们的使用许可，copyright 2002。

(续)

程序设计语言	LOC/FP			
	平均值	中值	低值	高值
JSP	59	—	—	—
Lotus Notes	23	21	15	46
Mantis	71	27	22	250
Natural	51	53	34	60
.NET	60	60	60	60
Oracle	42	29	12	217
OracleDev2K	35	30	23	100
PeopleSoft	37	32	34	40
Perl	57	57	45	60
PL/I	58	57	27	92
Powerbuilder	28	22	8	105
RPG II/III	61	49	24	155
SAS	50	35	33	49
Smalltalk	26	19	10	55
SQL	31	37	13	80
VBScript	38	37	29	50
Visual Basic	50	52	14	276

由这些数据可以看出，C++ 的一个 LOC 所提供的“功能”大约是 C 的一个 LOC 所提供功能的 2.4 倍（平均来讲）。Smalltalk 一个 LOC 所提供的“功能”至少是传统程序设计语言（如 Ada、COBOL 或 C）的 4 倍。利用上表包含的信息，只要知道了程序设计语言的语句行数，就可以“逆向”[Jon98] 估算出现有软件的功能点数量。

LOC 和 FP 测量经常用来导出生产率度量，这总会引起关于这些数据使用的争论。一个小组的 LOC/ 人月（或功能点 / 人月）应该与另一个小组的类似数据进行比较吗？管理者应该使用这些度量来评价个人绩效吗？对这些问题都斩钉截铁地回答“不！”原因是生产率受很多因素的影响，进行“苹果和橘子”式的比较很容易产生曲解。

人们发现，基于功能点的度量和基于 LOC 的度量都是对软件开发工作量和成本的比较精确的判定。为了使用 LOC 和 FP 进行估算（第 24 章），还必须建立一个历史信息基线。

在过程度量和项目度量中，主要应该关心生产率和质量——软件开发“输出量”（作为投入的工作量和时间的函数）的测量以及对生产的工作产品“适用性”的测量。为了进行过程改进和项目计划，必须掌握历史情况。在以往的项目中，软件开发的生产率是多少？生产的软件质量如何？怎样利用以往的生产率数据和质量数据推断现在的生产率和质量？如何利用这些数据帮助我们改进过程，以及更精确地规划新的项目？

23.2.4 面向对象的度量

传统的软件项目度量（LOC 或 FP）也可以用于估算面向对象的软件项目。但是，这些度量并没有提供对进度和工作量进行调整的足够的粒度，而这却是在演化模型或增量模型中进行迭代时所必需的。Lorenz 和 Kidd[Lor94] 提出了下列用于 OO 项目的度量。

场景脚本的数量。场景脚本（类似于用例）是一个详细的步骤序列，用来描述用户和应

用之间的交互。每个脚本采用如下三个一组的形式来组织：

{ 发起者, 动作, 参与者 }

其中, 发起者是指请求某个服务 (首先传递一个消息) 的对象, 动作是该请求的结果, 参与者是满足该请求的服务对象。场景脚本的数量与应用的规模及测试用例 (一旦构建出系统, 就必须设计测试用例来测试该系统) 的数量紧密相关。

关键类的数量。关键类是“高度独立的构件” [Lor94], 在面向对象分析的早期进行定义 (第 9 章)^①。由于关键类是问题域的核心, 因此, 这些类的数量既是开发软件所需工作量的指标, 也是系统开发中潜在的复用数量的指标。

支持类的数量。支持类是实现系统所必需的但又不与问题域直接相关的类。例如, 用户界面 (UI) 类、数据库访问及操作类、计算类。对每一个关键类, 都可以开发其支持类。在演化过程中, 支持类是迭代定义的。支持类的数量既是开发软件所需工作量的指标, 也是系统开发中潜在的复用数量的指标。

每个关键类的平均支持类数量。通常, 关键类在项目的早期就可以确定下来, 而支持类的定义则贯穿于项目的始终。对于给定的问题域, 如果知道了每个关键类的平均支持类数量, 估算 (根据类的总数) 就将变得极其简单。Lorenz 和 Kidd 指出, 在采用 GUI 的应用中, 支持类是关键类的 2 ~ 3 倍; 在不采用 GUI 的应用中, 支持类是关键类的 1 ~ 2 倍。

子系统的数量。子系统是实现某个功能 (对系统最终用户可见) 的类的集合。一旦确定了子系统, 人们就更容易制定出合理的进度计划, 并将子系统的工作在项目人员之间进行分配。

为了将上述这些度量有效地应用于面向对象的软件工程环境中, 必须将它们随同项目测量 (例如, 花费的工作量、发现的错误和缺陷、建立的模型或文档资料) 一起收集。随着数据库规模的增长 (在完成大量项目之后), 面向对象测量和项目测量之间的关系将提供有助于项目估算的度量。

23.2.5 面向用例的度量

用例^②被广泛地用于描述客户层或业务领域的需求, 这些需求中隐含着软件的特性和功能。与 LOC 或 FP 类似, 使用用例作为规范化的测量应该是合理的。用例同 FP 一样, 也是在软件过程早期进行定义。在重大的建模活动和构建活动开始之前, 就允许使用用例进行估算。用例描述了 (至少是间接地) 用户可见的功能和特性, 这些都是系统的基本需求。用例与程序设计语言无关。另外, 用例的数量同应用的规模 (LOC) 和测试用例的数量成正比, 而测试用例是为了充分测试该应用而必须要设计的。

由于可以在不同的抽象级别上创建用例, 所以用例的“大小”没有统一标准。由于对用例本身都没有标准的“测量”, 因此将用例作为规范化的测量 (例如, 每个用例花费的工作量) 是不可信的。

建议 多个场景脚本涉及同一个功能或数据对象, 这种情况很常见, 因此应该慎重使用脚本数量这个度量。有时可以将多个脚本简化为一个类或一组代码。

建议 每个类的规模和复杂性不同, 因此可以考虑将类按照规模和复杂性的不同分别统计其数量。

① 在第 9 章中, 关键类被称为分析类。

② 关于用例的介绍已在第 7 章和第 8 章中给出。

研究人员提议将用例点 (UCP) 作为一种估算项目工作量及其他特性的机制。UCP 是用例模型所包含的参与者数量及业务数量的函数,在某些方面与 FP 相似。如果有兴趣进一步了解,可参见 [Coh05]、[Cle06] 或 [Col09]。

23.3 软件质量的度量

系统、应用或产品的质量取决于描述问题的需求、建模解决方案的设计、导出可执行程序的编码以及执行软件来发现错误的测试。可以使用测量来获知需求与设计模型的质量、源代码的质量以及构建软件时所创建的测试用例的质量。为了做到这种实时的评价,必须应用产品度量来客观而不是主观地评估软件工作产品的质量。

随着项目的进展,项目经理也必须评估质量。将软件工程师个人收集的私有度量结合起来,可以提供项目级的结果。虽然可以收集到很多质量的测量数据,但在项目级上最主要的还是测量错误和缺陷。从这些测量中导出的度量能够提供一个指标,表明个人及小组在软件质量保证和控制活动上的效力。

度量,如每功能点的工作产品错误数、评审时每小时发现的错误数、测试时每小时发现的错误数,使我们能够深入了解度量所隐含的每项活动的功效。有关错误的数数据也能用来计算每个过程框架活动的缺陷排除效率 (Defect Removal Efficiency, DRE)。

23.3.1 测量质量

虽然有很多关于软件质量的测量指标,但正确性、可维护性、完整性和可用性为项目团队提供了有用的指标。Gilb[Gil88] 分别给出了它们的定义和测量。

正确性。正确性是软件完成所要求的功能的程度。缺陷 (正确性缺失) 是指在程序发布后经过了全面使用,由程序用户报告的问题。为了进行质量评估,缺陷是按标准时间段来计数的,典型的时间是一年。最常用的关于正确性的测量是每千行代码 (KLOC) 的缺陷数,这里的缺陷是指已被证实不符合需求的地方。

可维护性。可维护性是指遇到错误时程序能够被修改的容易程度,环境发生变化时程序能够适应的容易程度,以及用户希望变更需求时程序能够被增强的容易程度。还没有直接测量可维护性的方法,只能采用间接测量。有一种简单的面向时间的度量,称为平均变更时间 (Mean-Time-To-Change, MTTC)。平均变更时间包括分析变更请求、设计合适的修改方案、实现变更并进行测试以及把该变更发布给全部用户所花费的时间。

完整性。这个属性测量的是一个系统对安全性攻击 (包括偶然的和蓄意的) 的抵抗能力。为了测量完整性,必须定义另外两个属性:危险性和安全性。危险性是指一个特定类型的攻击在给定的时间内发生的概率 (能够估算或根据经验数据导出)。安全性是指一个特定类型的攻击被击退的概率 (能够估算出来或根据经验数据得到)。系统的完整性可以定义为:

$$\text{完整性} = \sum (1 - (\text{危险性} \times (1 - \text{安全性})))$$

例如,假设危险性 (发生攻击的可能性) 是 0.25,安全性 (击退攻击的可能性) 是 0.95,则系统的完整性是 0.99 (很高);另一方面,假设危险性是 0.5,击退攻击的可能性仅是 0.25,

建议 软件是一个复杂的实体。随着工作产品的开发,错误也会产生。过程度量就是要改进软件过程,以便更有效地发现错误。

网络资源 关于软件质量及相关主题 (包括度量) 的优秀信息源可参见 <http://searchsoftwarequality.techtarget.com/resources>。

则系统的完整性只有 0.63（低得无法接受）。

可用性。可用性力图对“使用的容易程度”进行量化，可以根据第 14 章中给出的特性来测量。

在被建议作为软件质量测量的众多因素中，上述 4 个因素仅仅是一个样本。

23.3.2 缺陷排除效率

缺陷排除效率（Defect Removal Efficiency, DRE）是在项目级和过程级都有意义的质量度量。质量保证及质量控制活动贯穿于所有过程框架活动中，DRE 本质上就是对质量保证及质量控制动作中滤除缺陷的能力的测量。

把项目作为一个整体来考虑时，可按如下方式定义 DRE：

$$DRE = \frac{E}{E+D}$$

其中， E 是软件交付给最终用户之前发现的错误数， D 是软件交付之后发现的缺陷数。

DRE 最理想的值是 1，即在软件中没有发现缺陷。实际上， D 的值大于 0，但 DRE 仍可能接近于 1。对于一个给定的 D 值，随着 E 的增加，DRE 的整体数值越来越接近于 1。实际上，随着 E 的增加， D 的最终值会降低（错误在变成缺陷之前已经被滤除了）。如果将 DRE 作为一个度量，提供关于质量控制及质量保证活动的滤除能力的衡量指标，那么 DRE 就能促使软件项目团队采用先进的技术，力求在软件交付之前发现尽可能多的错误。

建议 如果从分析阶段进入设计阶段时，DRE 的值较低，你就要花些时间去改进正式技术评审的方式了。

在项目内部，也可以使用 DRE 来评估一个团队在错误传递到下一个框架活动或软件工程任务之前发现错误的能力。例如，需求分析创建了一个需求模型，而且对该模型进行了评审来发现和改正其中的错误。那些在评审过程中未被发现的错误传递给了设计（在设计中它们可能被发现，也可能没有被发现）。在这种情况下，我们将 DRE 重新定义为：

$$DRE_i = \frac{E_i}{E_i + E_{i+1}}$$

其中， E_i 是在软件工程动作 i 中发现的错误数； E_{i+1} 是指在软件工程动作 $i+1$ 中发现的而在软件工程动作 i 中没有被发现的错误的数量。

软件团队（或软件工程师个人）的质量目标是使 DRE_i 接近于 1，即错误应该在传递到下一个活动或动作之前被滤除。

SafeHome 建立度量方法

[场景] Doug Miller 的办公室，在首次召开软件度量会议两天之后。

[人物] Doug Miller, SafeHome 软件团队经理；Vinod Raman 和 Jamie Lazar, 产品软件工程团队成员。

[对话]

Doug: 关于过程度量和项目度量，你们都有所了解了吧。

Vinod 和 Jamie 都点头。

Doug: 无论采用何种度量，都要建立目

标，这总是正确的。那么，你们的目标是什么？

Vinod：我们的度量应该关注质量。实际上，我们的总体目标是使得上一个软件工程活动传递给下一个软件工程活动的错误数最少。

Doug：并且确保随同产品发布的缺陷数尽可能接近于 0。

Vinod（点头）：当然。

Jamie：我喜欢将 DRE 作为一个度量。我认为我们可以将 DRE 用于整个项目的度量。同样，当从一个框架活动转到下一个框架活动时，也可以使用它。DRE 促使我们在每一步都去发现错误。

Vinod：我觉得还要收集我们用在评审上

的小时数。

Jamie：还有我们花费在每个软件工程任务上的总工作量。

Doug：可以计算出评审与开发的比率，这可能很有趣。

Jamie：我还想跟踪一些用例方面的数据，如建立一个用例所需的工作量，构建软件来实现一个用例所需的工作量，以及……

Doug（微笑）：我想我们要保持简单。

Vinod：应该这样，不过你一旦深入到度量中，就可以看到很多有趣的事。

Doug：我同意，但在我们会跑之前要先走，坚持我们的目标。收集的数据限制在 5 到 6 项，准备去做吧。

习题与思考题

- 23.1 用自己的话描述过程度量与项目度量之间的区别。
- 23.2 为什么有些软件度量是“私有的”？给出 3 个私有度量的例子，并给出 3 个公有度量的例子。
- 23.3 什么是间接测量？为什么在软件度量工作中经常用到这类测量？
- 23.4 Grady 提出了一组软件度量规则，你能在 23.1.1 节所列的规则中再增加 3 个规则吗？
- 23.5 产品交付之前，团队 A 在软件工程过程中发现了 342 个错误，团队 B 发现了 184 个错误。对于项目 A 和 B，还需要做什么额外的测量，才能确定哪个团队能够更有效地排除错误？你建议采用什么度量来帮助做出判定？哪些历史数据可能有用？
- 23.6 给出反对将代码行作为软件生产率度量的论据。当考虑几十个或几百个项目时，你说的情况还成立吗？
- 23.7 根据下面的信息域特性，计算项目的功能点值：
 - 用户输入数：23.
 - 用户输出数：60
 - 用户查询数：24
 - 文件数：8
 - 外部接口数：2
 假定所有的复杂度校正值都取“中等”值。
- 23.8 利用 23.2.3 节中给出的表格，基于每行代码具有的功能性，提出一个反对使用汇编语言的论据。再参考该表，讨论为什么 C++ 比 C 更好。
- 23.9 用于控制影印机的软件需要 23.00 行 C 语言代码和 4200 行 Smalltalk 语言代码。估算该影印机软件的功能点数。
- 23.10 某 Web 工程团队已经开发了一个包含 145 个网页的电子商务 WebApp。在这些页面中，有 65 个是动态页面，即根据最终用户的输入而在内部生成的页面。那么，该应用的定制指数是多少？
- 23.11 一个 WebApp 及其支持环境没有被充分地加强来抵御攻击。Web 工程师估计击退攻击的概率只

有 30%。系统不包含机密或有争议的信息，因此危险性概率只有 25%。那么，该 WebApp 的完整性是多少？

- 23.12 在一个项目结束时，确定在建模阶段发现了 30 个错误，在构建阶段发现了 12 个错误，这 12 个错误可以追溯到建模阶段没有发现的错误。那么，这两个阶段的 DRE 是多少？
- 23.13 软件团队将软件增量交付给最终用户。在第一个月的使用中，用户发现了 8 个缺陷。在交付之前，软件团队在正式技术评审和所有测试任务中发现了 242 个错误。那么在使用一个月之后，项目总的缺陷排除效率（DRE）是多少？

扩展阅读与信息资源

在过去的 20 年中，软件过程改进（SPI）受到了极大的关注。由于测量和软件度量是成功改进软件过程的关键，所以在很多 SPI 方面的书籍中也讨论度量。Arban（《Software Metrics and Software Methodology》，Wiley-IEEE Computer Society，2010）和 Rico（《ROI of Software Process Improvement》，J. Ross Publishing，2004）所著的书深入讨论了 SPI 以及能够帮助组织进行过程改进的度量。Ebert 及其同事（《Best Practices in Software Measurement》，Springer，2004）在 ISO 和 CMMI 标准的范畴内讨论了测量的使用。Kan（《Metrics and Models in Software Quality Engineering》，2nd ed., Addison-Wesley, 2002）介绍了相关度量的收集。

Ebert 和 Dumke（《Software Measurement》，Springer，2007）提供了将测量和度量应用于 IT 项目时的有用的处理措施。McGarry 及其同事（《Practical Software Measurement》，Addison-Wesley，2001）对评估软件过程提出了深层次的建议。Haug 及其同事已经编辑了一部值得收藏的论文集（《Software Process Improve：Metrics, Measurement, and Process Modeling》，Springer-Verlag，2001）。Florac 和 Carlton（《Measuring the Software Process》，Addison-Wesley，1999）以及 Fenton 和 Pfleeger（《Software Metrics: A Rigorous and Practical Approach》，Revised, Brooks/Cole Publishers, 1998）探讨了如何利用软件度量来取得改进软件过程的必要指标。

Wohlin 和他的同事（《Experimentation in Software Engineering》，Springer, 2012）讨论了用于分析软件过程的测量的使用方式。Jones（《Applied Software Measurement: Global Analysis of Productivity and Quality》，McGraw-Hill, 2008）、Laird 和 Brennan（《Software Measurement and Estimation》，Wiley-IEEE Computer Society Press, 2006）以及 Goodman（《Software Metrics：Best Practices for Successful IT Management》，Rothstein Associates, 2004）讨论了如何将软件度量用于项目管理和估算。Putnam 和 Myers（《Five Core Metrics》，Dorset House, 2003）利用一个包含 6000 多个软件项目的数据库，论证了如何使用 5 种核心度量——时间、工作量、规模、可靠性以及过程生产率——来控制软件项目。Maxwell（《Applied Statistics for Software Managers》，Prentice-Hall, 2003）给出了分析软件项目数据的技术。Munson（《Software Engineering Measurement》，Auerbach, 2003）讨论了大量的软件工程测量问题。Jones《Software Assessments, Benchmarks and Best Practices》，Addison-Wesley, 2000）描述了定量的测量以及定性的测量因素，帮助组织评估自身的软件过程和实践。

功能点测量已经成为一种广泛应用于软件工程工作很多领域的技术。国际功能点用户组出版了关于使用功能点度量的论文集（《The IFPUC Guide to IT and Software Measurement》，Auerbach, 2012）。Parthasarathy（《Practical Software Estimation：Function Point Methods for Insourced and Outsourced Projects》，Addison-Wesley, 2007）提供了综合性的指南。Garmus 和 Herron（《Function Point Analysis：Measurement Practices for Successful Software Projects》，Addison-Wesley, 2000）讨论了侧

重于功能点分析的过程度量。

关于 Web 工程工作的度量,已发表的资料比较少,不过 Clifton (《Advanced Web Metrics with Google Analytics》, 3rd ed., Sybex, 2012)、Kaushik (《Web Analytics 2.0 : Accountability and Science of Customer Centricity》, Sybex, 2009 和《Web Analytics : An Hour a Day》, Sybex, 2007)、Stern (《Web Metrics : Proven Methods for Measuring Web Site Success》, Wiley, 2002)、Inan 和 Kean (《Measuring the Success of Your Website》, Longman, 2002) 以及 Nobles 和 Grady (《Web Site Analysis and Reporting》, Premier Press, 2001) 等人的著作从商业和市场角度讨论了 Web 度量。

IEEE 总结了度量领域的最新研究 (《Symposium on Software Metrics》, 每年出版)。大量的关于过程度量和项目度量的信息源可以在网上获得。最新的参考文献参见 SEPA 网站 www.mhhe.com/pressman 下的 “software engineering resources”。

软件项目估算

要点浏览

概念: 软件的真实需求已经确定; 利益相关者都已到位; 软件工程师准备开始工作; 项目将要启动。但是如何进行下去呢? 软件项目计划包括 5 项主要活动——估算、进度安排、风险分析、质量管理计划和变更管理计划。在本章中, 我们只考虑估算——尝试确定构建一个特定的基于软件的系统或产品所需投入的资金、工作量、资源及时间。

人员: 软件项目经理——利用从项目利益相关者那里获得的信息以及从以往项目中收集的软件度量数据。

重要性: 你会在不知道要花多少钱、要完成多少任务以及完成工作需要多少时间的情况下建造房子吗? 当然不会。既然大多数基于计算机的系统和产品的成本大大超过建造一所大房子, 那么在开发

软件之前进行估算应该是合理的。

步骤: 估算首先要描述产品的范围, 然后将问题分解为一组较小的问题, 再以历史数据和经验为指南, 对每个小问题进行估算。在进行最终的估算之前, 要考虑问题的复杂度和风险。

工作产品: 生成一个简单的表, 描述要完成的任务和要实现的功能, 以及完成每一项所需的成本、工作量和时间。

质量保证措施: 这很困难。因为一直要等到项目完成时, 你才能真正知道。不过, 如果你有经验并遵循系统化的方法, 使用可靠的历史数据进行估算, 利用至少两种不同的方法创建估算数据点, 制定现实的进度表并随着项目的进展不断进行调整, 那么你就可以确信你已经为项目做了最好的估算。

软件项目管理从一组统称为项目计划的活动开始。在项目启动之前, 软件团队应该估算将要的工作、所需的资源, 从开始到完成所需要的时间。这些活动一旦完成, 软件团队就应该制定项目进度计划。在项目进度计划中, 要定义软件工程任务及里程碑, 指定每一项任务的负责人, 详细说明对项目进展有较大影响的任务间的相互依赖关系。

在一本关于“软件项目生存”的优秀指南中, Steve McConnell [McC98] 讲述了人们对项目计划的实际看法:

很多技术工作者宁愿从事技术工作, 也不愿花费时间制定计划。很多技术管理者没有受过充分的技术管理方面的培训, 对他们的计划是否能够改善项目成果缺乏信心。既然这两部分人都不想制定计划, 因此就经常不制定计划。

关键概念

估算

敏捷开发

分解技术

经验模型

基于功能点

面向对象的

项目

基于问题

基于过程

估算

调和

用例

但是,没有很好地制定计划是一个项目犯的最严重的错误之一……有效的计划是必需的,可以在上游(项目早期)以较低的成本解决问题,而不是在下游(项目后期)以较高的成本解决问题。一般的项目要将80%的时间花费在返工上——改正项目前期所犯的错误。

McConnell指出,每个项目都能找出制定计划的时间(并使计划适应于整个项目),只要从因为没制定计划而出现的返工时间中抽出一小部分时间即可。

24.1 对估算的观察

制定计划需要你做一个初始约定,即使这个“约定”很可能被证明是错误的。无论在什么时候进行估算,都是在预测未来,自然要接受一定程度的不确定性。下面引用 Frederick Brooks[Bro95]的话:

……我们的估算技术发展缓慢。更为严重的是,它们隐含了一个很不正确的假设,即“一切都会好的”……因为对自己的估算没有把握,软件管理者常常缺乏做出好产品的信心。

估算是一门艺术,更是一门科学,这项重要的活动不能以随意的方式进行。现在已经有了估算时间和工作量的实用技术。过程度量和项目度量为定量估算提供了历史依据和有效输入。当建立估算和评审估算时,以往的经验(包括所有参与人员的经验)具有不可估量的辅助作用。由于估算是所有项目计划活动的基础,而项目计划提供了通往成功的软件工程的路线图。因此,没有估算就着手开发将会使我们陷入盲目。

对软件工程工作的资源、成本及进度进行估算时,需要经验,需要了解有用的历史信息(度量)。当只存在定性的信息时,还要有进行定量预言的勇气。估算具有与生俱来的风险[⊖],正是这种风险导致了不确定性。

项目的复杂性对计划固有的不确定性有很大影响。但是,复杂性是一个相对量,受人员在以往工作中对它的熟悉程度的影响。一个高级的电子商务应用对于首次承担开发工作的程序员来说可能极其复杂。但是,当Web工程团队第十次开发电子商务WebApp时,会认为这样的工作很普通。现在已经提出了很多软件复杂性的定量测量方法[Zus97]。这些测量方法都应用于设计层或代码层,因此在软件策划(先于设计和代码的存在)期间难以使用。但是,其他更主观的复杂性评估方法(例如,功能点复杂度校正系数)可以在早期的策划过程中建立。

项目规模是另一个能影响估算精确度和功效的重要因素。随着规模的扩大,软件各个元素之间的相互依赖迅速上升[⊖]。问题分解作为一种重要的估算方法,会由于问题元素的细化仍然难以完成而变得更加困难。用Murphy法则来解释:“凡事只要有可能出错,那就一定会出错。”这意味着如果有更多事情可能失败,则这些事情一定失败。

关键概念

项目计划
资源
软件方程
软件范围
软件规模估算
用例点(UCP)

引述 良好的估算方法和可靠的历史数据提供了最好的希望:现实将战胜不可能的要求。

Caper Jones

关键点 项目复杂性、项目规模以及结构的不确定程度都影响着估算的可靠性。

⊖ 系统的风险分析技术将在第26章讨论。

⊖ 当问题的需求改变时,由于“范围的蔓延”使得问题的规模常常会扩大。而项目规模的扩大对项目的成本和进度有几何级数的影响(Michael Mah, personal communication)。

结构的不确定程度也影响着估算的风险。这里，结构是指需求已经被固化的程度、功能被划分的容易程度以及必须处理的信息的层次特性。

历史信息的有效性对估算的风险有很大影响。通过回顾过去，你能仿效做过的工作，并改进出现问题的地方。如果能取得以往项目的全面软件度量（第 23 章），估算会有更大的保证，合理安排进度以避免重走过去的弯路，总体风险就会降低。

估算的风险取决于资源、成本及进度的定量估算中存在的 uncertainty。如果对项目范围不够了解，或者项目需求经常改变，不确定性和估算风险就会非常高。作为计划人员，你和客户都应该认识到经常改变软件需求意味着成本和进度上的不稳定性。

不过，你不应该被估算所困扰。现代软件工程方法（例如，演化过程模型）采用迭代开发方法。在这类方法中，当客户改变需求时，应该能够（尽管并不总是易于接受）重新审查估算（在了解更多信息后），并进行修正。

引述 应该满足于事物本性所能容许的精确度，当只可能近似于准确时，不要去寻求绝对的准确。
Aristotle

24.2 项目计划过程

软件项目计划的目标是提供一个能使管理人员对资源、成本及进度做出合理估算的框架。此外，估算应该尝试定义“最好的情况”和“最坏的情况”，使项目的结果能够限制在一定范围内。项目计划是在计划任务中创建的，尽管它具有与生俱来的不确定性，软件团队还是要根据它来着手开发。因此，随着项目的进展，必须不断地对计划进行调整和更新。在下面几节中，将讨论与软件项目计划有关的每一项活动。

建议 你了解的越多，就估算得越好。因此，随着项目的进展要对估算进行更新。

任务集 项目计划任务集

1. 规定项目范围
2. 确定可行性
3. 分析风险（第 26 章）
4. 确定需要的资源
 - a. 确定需要的人力资源
 - b. 确定可复用的软件资源
 - c. 识别环境资源
5. 估算成本和工作量
 - a. 分解问题
 - b. 使用规模、功能点、过程任务或用例等方法进行两种以上的估算。
 - c. 调和不同的估算
6. 制定项目进度计划（第 25 章）
 - a. 建立一组有意义的任务集合
 - b. 定义任务网络
 - c. 使用进度计划工具制定时间表
 - d. 定义进度跟踪机制

24.3 软件范围和可行性

软件范围描述了将要交付给最终用户的功能和特性、输入和输出数据、作为使用软件的结果呈现给用户的“内容”，还界定了系统的性能、约束条件、接口和可靠性。定义范围可以使用两种技术：

1. 在与所有利益相关者交流之后，写出软件范围的叙述性描述。
2. 由最终用户开发的一组用例。^①

在开始估算之前，首先要对范围陈述（或用例）中描述的功能进行评估，在某些情况下，还要进行细化，以提供更多的细节。由于成本和进度的估算都是面向功能的，因此一定程度的功能分解常常是有益的。性能方面的考虑包括处理时间和响应时间的需求。约束条件表示外部硬件、可用存储或其他现有系统对软件的限制。

一旦确定了软件范围（并征得用户的同意），人们自然会问：我们能够开发出满足范围要求的软件吗？这个项目可行吗？软件工程师常常匆忙越过这些问题（或是被不耐烦的管理者或其他利益相关者催促着越过这些问题），不料竟会一开始就注定要陷入这个项目的泥潭中。

Putnam 和 Myers 建议，只确定范围还不够。一旦理解了范围，就必须进一步确定在可用的技术、资金、时间和资源的框架下，所标识的范围是否能够完成。这是估算过程至关重要的部分，但常常被忽略。

建议 项目可行性很重要，但是，考虑商业需要更重要。构建一个没人想要的高科技系统或产品根本没有意义。

24.4 资源

项目计划的第二个任务是对完成软件开发工作所需的资源进行估算。图 24-1 描述了三类主要的软件工程资源——人员、可复用的软件构件及开发环境（硬件和软件工具）。对每类资源，都要说明以下四个特征：资源描述、可用性说明、何时需要资源以及使用资源的持续时间。最后两个特性可以看成是时间窗口。对于一个特定的时间窗口，必须在最早的使用时间建立资源的可用性。

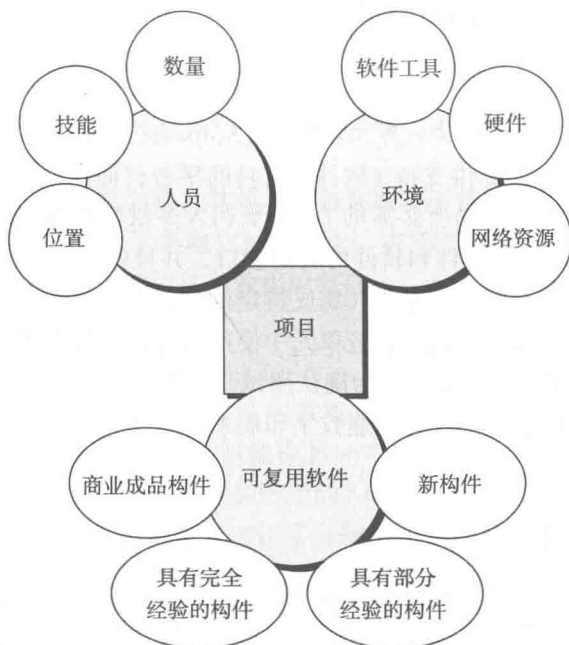


图 24-1 项目资源

^① 在本书的第二部分中，已经对用例进行了详细的讨论。用例从用户的观点出发来描述用户与软件的交互场景。

24.4.1 人力资源

计划人员首先评估软件范围,选择完成开发所需的技能,还要指定组织中的职位(例如,管理人员、高级软件工程师)和专业(例如,电信、数据库、客户/服务器系统)。对于一些比较小的项目(几个人月),只要向专家做些咨询,或许一个人就可以完成所有的软件工程任务。而对于一些较大的项目,软件团队的成员可能分散在多个不同的地方,因此,要详细说明每个人所处的位置。

只有在估算出开发工作量(如多少人月)后,才能确定软件项目需要的人员数量。估算工作量的技术将在本章后面讨论。

24.4.2 可复用软件资源

基于构件的软件工程(CBSE)^①强调可复用性,即创建并复用软件构件块,这种构件块通常称为构件。为了容易引用,必须对这些构件进行分类;为了容易应用,必须使这些构件标准化;为了容易集成,必须对这些构件进行确认。Bennatan[Ben00]建议在制定计划时应该考虑四种软件资源:成品构件(能够从第三方或者从以往项目中获得的现成软件),具有完全经验的构件(为以前项目开发的,具有与当前项目要构建的软件类似的规格说明、设计、代码或测试数据)、具有部分经验的构件(为以前项目开发的,具有与当前项目要构建的软件相关的规格说明、设计、代码或测试数据,但是需要做很多修改),以及新构件(软件团队为了满足当前项目的特定要求,专门开发的软件构件)。

具有讽刺意味的是,在计划阶段,人们往往忽视可复用软件构件,直到软件过程的开发阶段,它才变成最重要的关注对象。最好尽早确定软件资源的需求,这样才能对各种候选方案进行技术评估,及时获取所需的构件。

24.4.3 环境资源

支持软件项目的环境通常称为软件工程环境(Software Engineering Environment, SEE),它集成了硬件和软件。硬件提供支持(软件)工具的平台,而这些(软件)工具是采用良好的软件工程实践来获得工作产品所必需的^②。由于在大多数软件组织中,很多人都需要使用SEE,因此必须详细规定需要硬件和软件的时间窗口,并且验证这些资源是可用的。

当软件团队构建基于计算机的系统(集成特定的硬件和软件)时,可能需要使用其他工程团队开发的硬件元素。例如,在为制造单元中使用的机器人装置开发软件时,可能需要特定的机器人(例如,机器人焊接工)作为确认测试步骤的一部分;在开发高级排版软件项目的过程中,在某个阶段可能需要一套高速数字印刷系统。作为计划的一部分,必须指定每一个硬件元素。

24.5 软件项目估算

软件的成本及工作量估算从来都没有成为一门精确的科学。因为变化的因素太多——人员、技术、环境和行政,都会影响软件的最终成本和开发所用的工作量。不过,软件项目估算还是能够从一种“神秘的”技巧变成一系列系统化的步骤,在可接受的风险范围内提供估

^① 第13章讨论了CBSE。

^② 其他硬件(目标环境)是指在软件交付给最终用户之后,将要运行该软件的计算机。

算结果。为得到可靠的成本和工作量估算，我们有很多选择：

1. 把估算推迟到项目的后期进行（显然，在项目完成之后就能得到 100% 精确的估算）。
2. 根据已经完成的类似项目进行估算。
3. 使用比较简单的分解技术，生成项目的成本和工作量估算。
4. 使用一个或多个经验模型来进行软件成本和工作量的估算。

遗憾的是，不论多吸引人，第一种选择都是不现实的。成本估算必须“预先”给出。不过，应该认识到，你等待的时间越久，了解的就越多。而了解的越多，在估算中出现严重错误的可能性就越小。

如果当前项目与以前的工作非常相似，并且项目的其他影响因素（例如，客户、商业条件、软件工程环境、交付期限）也大致相同，第二种选择就能很好地发挥作用。遗憾的是，过去的经验并不总是能够指明未来的结果。

余下的两种选择对于软件项目估算也是可行的方法。理想的情况是，同时使用这两种选项所提到的技术，相互进行交叉检查。分解技术采用“分而治之”的方法进行软件项目估算，把项目分解成若干主要功能和相关的软件工程活动，以逐步求精的方式对成本和工作量进行估算。经验估算模型可以作为分解技术的补充，在它适用的范围内常常是一种有潜在价值的估算方法。一个基于经验（历史数据）的模型形式如下：

$$d = f(v_i)$$

其中， d 是很多估算值（例如，工作量、成本、项目持续时间）中的一种， v_i 是所选的独立参数（例如，被估算的 LOC 或 FP）。

自动的估算工具实现了一种或多种分解技术或经验模型，提供了有吸引力的估算选择。使用这样的系统进行估算时，要描述开发组织的特性（如经验、环境）和待开发的软件，再由这些数据导出成本和工作量估算。

对于每种可行的软件成本估算方法，其效果的好坏取决于估算所使用的历史数据。如果没有历史数据，成本估算就建立在了不稳定的基础上。在第 23 章中，我们已经考察了一些软件度量的特性，这些度量提供了历史估算数据的基础。

24.6 分解技术

软件项目估算是解决问题的一种方式，在多数情况下，要解决的问题（对于软件项目来说，就是成本和工作量估算）非常复杂，不能作为一个整体考虑。因此，要对问题进行分解，把它分解成一组较小的（同时有望更容易管理的）问题，再定义它们的特性。

在第 22 章中，我们从两个不同的角度讨论了分解方法：问题分解和过程分解。估算时可以使用其中一种或两种分解形式。但在进行估算之前，必须理解待开发软件的范围，并估计其“规模”。

24.6.1 软件规模估算

软件项目估算的准确性取决于许多因素：（1）估算待开发产品的规模的正确程度；（2）把规模估算转换成人员工作量、时间及成本的能力（受可靠软件度量的可用性的影响，

引述 在外包和竞争愈加激烈的时代，更准确地进行估算的能力……已经成为很多 IT 组织成功的关键因素。

Rob Thomsett

引述 不采用定量的方法、几乎没有数据支持、主要由管理人员的直觉来保证，这样就很难制定有效、可靠、防御工作风险的估算。

Fred Brooks

这些度量数据来自以往的项目)；(3) 项目计划反映软件团队能力的程度；(4) 产品需求的稳定性和支持软件工程工作的环境。

由于项目估算的准确程度取决于待完成工作的规模估算，因此规模估算是计划人员面临的第一个主要挑战。在项目计划中，规模是指软件项目的可量化结果。如果采用直接的方法，规模可以用代码行 (LOC) 来测量。如果选择间接的方法，规模可以用功能点 (FP) 来表示。规模的估算要考虑项目类型以及应用领域类型、要交付的功能（如功能点数量）、要交付的构件数量、对现有构件做适用于新系统的修改的程度。

Putnam 和 Myers 建议，可以将上述每种规模估算方法所产生的结果在统计意义上结合起来，产生一个三点估算或期望值估算结果。实现方法是：先确定规模的乐观值（低）、可能值和悲观值（高），然后使用 24.6.2 节的式 (24.1) 将它们结合起来。

24.6.2 基于问题的估算

在第 32 章中，已经描述了代码行和功能点测量，从中可以计算出生产率度量。在软件项目估算中，LOC 和 FP 数据用于两个方面：(1) 作为估算变量，度量软件中每个元素的规模；(2) 作为基线度量，这些度量数据是从以前的项目中收集起来的，将它们与估算变量结合使用，进行成本和工作量的估算。

LOC 估算和 FP 估算是两种不同的估算技术，但两者有很多相同特性。首先从界定的软件范围陈述入手，尝试将范围陈述分解成一些可分别独立进行估算的功能问题。然后，估算每个功能的 LOC 或 FP（即估算变量）。当然，也可以选择其他元素进行规模估算，例如，类或对象、变更、受影响的业务过程。

然后，将基线生产率度量（例如，LOC/pm 或 FP/pm^①）应用于适当的估算变量，导出每个功能的成本或工作量。将所有功能的估算合并起来，即可得到整个项目的总体估算。

然而，应该注意到，对于一个组织而言，其生产率度量常常是变化的，使用单一基线的生产率度量并不可信。一般情况下，平均 LOC/pm 或 FP/pm 应该根据项目领域来计算。即应该根据项目的团队规模、应用领域、复杂性以及其他相关参数对项目进行分类，然后计算出本领域的生产率平均值。估算一个新项目时，首先应将项目对应到某个领域中，然后再使用适当的领域生产率平均值对其进行估算。

在分解应用问题时，LOC 估算技术和 FP 估算技术所要求的详细程度及划分目标有所不同。当 LOC 用作估算变量时，分解是绝对必要的，而且常常要达到非常详细的程度。分解的程度越高，就越有可能得到非常精确的 LOC 估算。

对于 FP 估算，分解则是不同的。它关注的不是功能，而是 5 个信息域特性——输入、输出、数据文件、查询和外部接口，以及 14 种复杂度校正值。然后，利用这些估算结果导出 FP 值，该值可与过去的数据结合起来产生估算。

不管使用哪一种估算变量，你都应该首先为每个功能或每个信息域值确定一个估算值的

关键点 待开发

软件的规模可以使用直接测量 LOC 或间接测量 FP 来估算。

提问 基于 LOC

和基于 FP 的估算有什么共同点？

建议 在收集项

目的生产率度量时，一定要划分项目类型。这样才能计算出特定领域的平均值，从而使估算更精确。

^① 缩写 pm 代表工作量的单位——人月 (person-month)。

范围。利用历史数据或凭直觉（当其他方法都失效时），为每个功能或每个信息域的计数值都分别估算出一个乐观的、可能的和悲观的规模值。在确定了值的范围后，就得到了一个不确定程度的隐含指标。

接着，计算三点（估算值）或期望值。可以通过乐观值（ S_{opt} ）、可能值（ S_{m} ）和悲观值（ S_{pess} ）估算的加权平均值来计算估算变量（规模） S 的期望值，例如：

$$S = \frac{S_{\text{opt}} + 4S_{\text{m}} + S_{\text{pess}}}{6} \quad (24.1)$$

其中，“可能”估算值的权重最大，并遵循 β 概率分布。我们假定实际的规模结果落在乐观值与悲观值范围之外的概率很小。

一旦确定了估算变量的期望值，就可以应用历史的 LOC 或 FP 生产率数据。这个估算正确吗？对于这个问题唯一合理的答案就是：“我们不能保证”。对于任何估算技术，不管它有多先进，都必须与其他方法进行交叉检查。即便如此，常识和经验还是会占优势。

24.6.3 基于 LOC 估算的实例

作为 LOC 和 FP 基于问题估算技术的实例，我们考虑为机械零件计算机辅助设计（CAD）应用开发的软件包。该软件将在桌面工作站上运行，且必须与各种计算机外部绘图设备有接口，包括鼠标、数字化仪、高分辨率彩色显示器和激光打印机。可以给出初步的软件范围陈述：

机械 CAD 软件接受工程师输入的二维或三维几何数据。工程师通过用户界面与 CAD 系统进行交互并控制它，该用户界面应表现出良好的人机界面设计特征。所有的几何数据和其他支持信息都保存在一个 CAD 数据库中。要开发一些设计分析模块，以产生所需的输出，这些输出要显示在各种不同的设备上。软件必须能够控制外部设备（包括鼠标、扫描仪、激光打印机和绘图机），并能与外部设备进行交互。

上述关于范围的陈述是初步的——它没有规定边界。必须对每个句子进行补充说明，以提供具体的细节及定量的边界。例如，在开始估算之前，计划人员必须要确定“良好的人机界面设计特征”是什么含义，或“CAD 数据库”的规模和复杂度是怎样的。

假定我们为了进行估算已经做了进一步的细化，确定了该软件包应具有的主要软件功能，如图 24-2 中所示。遵照 LOC 的分解技术，得到如图 24-2 所示的估算表，表中给出了每个功能的 LOC 估算范围。例如，三维几何分析功能的 LOC 估算范围是：乐观值 4600，可能值 6900，悲观值 8600。应用式（24.1），得到三维几何分析功能的期望值是 6800LOC。通过类似的方法也可以得到其他估算。对 LOC 估算这一列求和，就得到了该 CAD 系统的 LOC 估算值是 33200。

回顾历史数据可以看出，这类系统的组织平均生产率是 620LOC/pm。如果一个劳动力的价格是每月 8000 美元，则每行代码的成本约为 13 美元。根据 LOC 估算及历史生产率数据，该项目总成本的估算值是 431000 美元，工作量的估算值是 54 人月^①。

提问 我们如何计算软件规模的“期望值”？

建议 很多现代应用或者驻留在网络上，或者是客户机/服务器体系结构的一部分。因此，要确信你的估算包含了开发“基础设施软件”所需的工作量。

建议 不要屈服于诱惑而使用某一结果作为你的项目估算结果。应该再使用其他方法导出另一个结果来。

① 估算单位是千美元和人月。如果给定了估算的精确度要求，则更高的精确度是不必要的，也是不现实的。

功能	LOC 估算
用户接口及控制设备 (UICF)	2300
二维几何分析 (2DGA)	5300
三维几何分析 (3DGA)	6800
数据库管理 (DBM)	3350
计算机图形显示设备 (CGDF)	4950
外部设备控制功能 (PCF)	2100
设计分析模块 (DAM)	8400
总代码行估算	33200

图 24-2 LOC 方法的估算表

SafeHome | 估算

[场景] 项目计划开始时, Doug Miller 的办公室。

[人物] Doug Miller, SafeHome 软件团队经理; Vinod Raman、Jamie Lazar 及其他产品软件工程团队成员。

[对话]

Doug: 我们需要对这个项目进行工作量估算, 然后还要为第一个增量制定微观进度计划, 为其余的增量制定宏观进度计划。

Vinod (点头): 好, 但是我们还没有定义任何增量。

Doug: 是的, 但这正是我们需要估算的原因。

Jamie (皱着眉): 你想知道这将花费我们多长时间吗?

Doug: 这正是我需要的。首先, 我们要对 SafeHome 软件进行高层次上的功能分解, 接着, 我们必须估算每个功能对应的代码行数, 然后……

Jamie: 等一下! 我们应该怎样做来?

Vinod: 在过去的项目中, 我已经做过了。你从用例入手, 确定实现每个用例所需要的功能, 然后估计每项功能的 LOC 数。最好的方法是让每个人独立去做, 然后比较结果。

Doug: 或者你可以对整个项目进行功能分解。

Jamie: 但那将花费很长时间, 而我们必须马上开始。

Vinod: 不, 事实上这可以在几个小时内完成, 就今天早上。

Doug: 我同意, 我们不能期望有很高的精确性, 只是了解一下 SafeHome 软件的大致规模。

Jamie: 我认为我们应该只估算工作量, 仅此而已。

Doug: 这我们也要做。然后用这两种估算进行交叉检查。

Vinod: 我们现在就去做吧……

24.6.4 基于 FP 估算的实例

基于 FP 估算时, 问题分解关注的不是软件功能, 而是信息域的值。分别对 CAD 软件的输入、输出、查询、文件和外部接口进行估算, 参看图 24-3 给出的表。然后计算 FP 的值。为了进行估算, 假定复杂度加权因子都取平均值。图 24-3 给出了估算的结果。

信息域值	乐观值	可能值	悲观值	估算值	加权因子	FP 值
外部输入数	20	24	30	24	4	97
外部输出数	12	15	22	16	5	78
外部查询数	16	22	28	22	5	88
内部逻辑文件数	4	4	5	4	10	42
外部接口文件数	2	2	3	2	7	15
总计						320

图 24-3 估算信息域的值

估算出每一个复杂度加权因子，计算出复杂度校正因子的值：

因子	值	因子	值
备份和恢复	4	信息域值复杂度	5
数据通信	2	内部处理复杂度	5
分布式处理	0	设计可复用的代码	4
关键性能	4	设计中的转换与安装	3
现有的操作环境	3	多次安装	5
联机数据输入	4	易于变更的应用设计	5
多屏幕输入切换	5	复杂度校正因子	1.17
主文件联机更新	3		

最后，得出 FP 的估算值：

$$FP_{\text{estimated}} = \text{总计} \times (0.65 + 0.01 \times \sum F_i) = 375$$

这类系统的组织平均生产率是 6.5FP/pm。如果一个劳动力价格是每月 8000 美元，则每个 FP 的成本约为 1230 美元。根据 FP 估算和历史生产率数据，项目总成本的估算值是 461000 美元，工作量的估算值是 58 人月。

24.6.5 基于过程的估算

最通用的项目估算技术是根据将要采用的过程进行估算，即将过程分解为一组较小的活动、动作和任务，并估算完成每一项所需的工作量。

同基于问题的估算技术一样，基于过程的估算首先从项目范围中抽取软件功能。接着给出为实现每个功能所必须执行的一系列框架活动。这些功能及其相关的框架活动^①可以用表格形式给出，类似于图 24-4 所示。

一旦将问题功能与过程活动结合起来，就可以针对每个软件功能，估算出完成各个软件过程活动所需的工作量（如人月），这些数据构成了图 24-4 中表格的中心部分。然后，将平均劳动力价格（即成本/单位工作量）应用于每个软件过程活动的估算工作量，就可以估算出成本。

建议 如果时间允许，可以使用更细的粒度来指定图 24-4 中所示的任务，例如，将分析分解为若干主要任务，并分别估算每一项任务。

① 为该项目选择的框架活动与第 3 章中讨论的一般性活动有所不同。这些框架活动是客户沟通（CC）、策划、风险分析、工程和构建/发布。

活动→	客户沟通	策划	风险分析	工程		构建发布		客户评估	合计
任务→				分析	设计	编码	测试		
功能									
↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
合计	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% 工作量	1%	1%	1%	8%	45%	10%	36%		

图 24-4 基于过程的估算表

如果基于过程的估算是 不依赖 LOC 或 FP 估算而实现的，那么就 已经有了两种或三种成本与工作量估算，可以进 行比较和调和。如果两组 估算非常一致，则有理由 相信估算是可靠的。相反，如果这些分解技术得到的结果不一致，则必须做进一步的调查和分析。

24.6.6 基于过程估算的实例

为了说明基于过程估算的使用方法，我们再次考虑 24.6.3 节介绍的 CAD 软件。系统配置和所有软件功能都保持不变，并已在项目范围中说明。

参看图 24-4 中所示的基于过程的估算表，表中对 CAD 软件的每个功能（为了简化做了省略）都给出了其各个软件工程活动的工作量估算（人月）。其中，工程和构建发布活动又被细分为主要的软件工程任务。对客户沟通、策划和风险分析活动，还给出了总工作量的估算，这些数值都列在表格底部的“合计”行中。水平合计和垂直合计为估算分析、设计、编码及测试所需的工作量提供了指标。应该注意到，“前期”的工程任务（需求分析和设计）花费了全部工作量的 53%，说明这些工作相对更重要。

如果平均一个劳动力的价格是每月 8000 美元，则项目总成本的估算值是 368000 美元，工作量的估算值是 46 人月。如果需要的话，每个框架活动或软件工程任务都可以采用不同的劳动力价格分别进行计算。

24.6.7 基于用例的估算

正如本书第二部分中提到的，用例能使软件团队深入地 了解软件的范围和需求。一旦开发出用例，就能够将其用于 估算软件项目的计划“规模”。但是，建立基于用例的估算方法还面临着挑战 [Smi99]。用例的描述

引述 在使用一种估算之前，最好先去了解这种估算的背景。
Barry Boehm,
Richard Fairley

提问 为什么开发基于用例的估算技术非常困难？

可以采用多种格式和风格,用例表现的是软件的外部视图(用户视图),因此可以在多个不同的抽象级别上建立用例。用例没有标识出它所描述的功能和特性的复杂性,用例不能描述涉及很多功能和特性的复杂行为(如交互)。

尽管有这些限制,但使用类似于功能点计算的方式来计算用例点(Use Case Point, UCP)还是可能的。

Cohn(Coh05)指出,用例点的计算必须考虑以下特性:

- 系统中用例的数量和复杂性。
- 系统参与者的数量和复杂性。
- 没有写成用例的各种非功能性需求(如可移植性、性能、可维护性)。
- 项目的开发环境(如编程语言、软件团队的积极性)。

首先,评估每个用例,确定其相对复杂性。简单的用例预示着简单的用户界面、单一的数据库、3个以下的事务以及可用5个以下的类实现。一般性的用例预示着比较复杂的用户界面、2或3个数据库,以及4到7个事务含有5到10个类。最后,复杂的用例预示着复杂的用户界面,有多个数据库,使用8个以上的事务以及11个以上的类。采用这些标准评估每一个用例,将每种类型的用例数量分别乘以一个加权系数5、10或15。将加权后的用例数量求和得到总体的未调整用例权重(Unadjusted Use Case Weight, UUCW) [Nun11]。

接着,评估每个参与者。简单的参与者是一个通过API进行通信的自动机(另一个系统、一台机器或设备)。一般性的参与者是一个通过协议或数据存储来通信的自动机。复杂的参与者是人,通过图形用户界面或其他人机接口进行通信。使用这些标准评估每个参与者,将每种类型的参与者的数量分别乘以一个加权系数1、2或3,将加权后的参与者数量求和得到总体的未调整的参与者权重(Unadjusted Actor Weight, UAW)。

考虑技术复杂性因子(TCF)和环境复杂性因子(ECF),对这些未调整值进行修改。有13个因子用于估算最终的TCF,8个因子用于最终的ECF的计算[Coh05]。一旦确定了这些值,则以下面的方式来计算最终的用户点值:

$$UCP = (UUCW + UAW) \times TCF \times ECF \quad (24.2)$$

24.6.8 基于用例点估算的实例

在24.6.3节介绍的CAD软件包括3个子系统组:用户界面子系统(包括UICF)、工程子系统组(包括2DGA、3DGA和DAM子系统)及基础设施子系统组(包括CGDF子系统和PCF子系统)。用户界面子系统由16个复杂用例描述。工程子系统组由14个一般用例和8个简单用例描述。基础设施子系统由10个简单用例描述。因此,

$$UUCW = (16 \text{ 用例} \times 15) + ((14 \text{ 用例} \times 10) + (8 \text{ 用例} \times 5)) + (10 \text{ 用例} \times 5) = 470$$

对用例进行分析可发现有8个简单的参与者,12个一般的参与者,4个复杂的参与者。因此,

$$UAW = (8 \text{ 参与者} \times 1) + (12 \text{ 参与者} \times 2) + (4 \text{ 参与者} \times 3) = 44$$

对技术和环境进行评估后,确定TCF和ECF的值:

$$TCF = 1.04 \quad ECF = 0.96$$

利用式(24.2)可得:

$$UCP = (470 + 44) \times 1.04 \times 0.96 = 513$$

以过去的项目数据为依据,开发小组每个UCP生产85 LOC。这样,CAD项目的总体

规模估计为 43600LOC。对投入的工作量或者项目工期也可以做类似的计算。

以 620LOC/pm 作为这类系统的平均生产率,一个劳动力价格是每月 8000 美元,则每行代码的成本约为 13 美元。根据用例估算和历史生产率数据,项目总成本的估算值是 552000 美元,工作量的估算值大约是 70 人月。

24.6.9 调和不同的估算方法

在前面章节中讨论的估算技术导出了多种估算方法,必须对这些估算方法进行调和,以得到对工作量、项目工期或成本的一致估算。对 CAD 软件(24.6.3 节)总工作量的估算,最低值是 46 人月(由基于过程的估算方法得出),最高值是 68 人月(由用例估算方法得出)。平均估算值(使用全部 4 种方法)是 56 人月。与平均估算值相比,最低估算值的偏差约为 18%,最高估算值的偏差约为 21%。

如果估算方法所得结果的一致性很差,怎么办呢?对这个问题的回答是,需要对估算所使用的信息进行重新评估。如果不同估算之间的差别很大,一般能够追溯到以下两个原因之一:(1)计划人员没有充分理解或是误解了项目范围;(2)在基于问题的估算技术中所使用的生产率数据不适合本应用,它们已经过时了(因为这些数据已不能正确反映软件工程组织的情况),或者是误用了。应该确定产生差别的原因,再来调和估算结果。

引述 复杂的方法并不一定会产生更精确的估算,尤其是当开发者在估算时加进自己的直觉时。

Philip Johnson
et al.

信息栏 软件项目的自动估算技术

自动估算工具允许计划人员估算成本和工作量,还可以对重要的项目变量(例如,交付日期或人员配置)进行假设分析。尽管目前已有很多自动估算工具(参看本章后面的补充材料),但它们具有相同的本质特性,都能实现以下 6 项一般性功能 [Jon96]。

1. 估算项目可交付产品的规模——估算一个或多个软件工作产品的“规模”。工作产品包括软件的外部表示(如屏幕、报表)、软件本身(如 KLOC)、交付的功能(如功能点)及描述性信息(如文档)。
2. 选择项目活动——选择适当的过程框架,指定软件工程任务集。
3. 预测人员配置标准——指定可用的工作人员数量。由于可用人员和工作(预测的工作量)之间完全是非线性关系,因此这是一项重要输入。
4. 预测软件工作量——估算工具使用一个或多个估算模型(24.7 节)来预测软件工作量,这些模型能将交付的项目规模与开发所需的工作量联系起来。
5. 预测软件成本——如果给出第 4 步的结果,再分别指定第 2 步中确定的项目活动的劳动力价格,就可以计算出成本。
6. 预测软件进度计划——当工作量、人员配置和项目活动已知后,可以根据本章后面讨论的工作量分配推荐模型,来分配各个软件工程活动的人员,从而制定出进度计划草案。

当把不同的估算工具应用于相同的项目数据时,估算结果会有比较大的变动范围。更重要的是,有时候预测值会与实际值差异很大。这再次证明了应该把估算工具的输出看作是一个“数据点”,再从中导出估算,而不是将其作为估算的唯一来源。

24.7 经验估算模型

计算机软件估算模型使用由经验导出的公式来预测工作量，工作量是 LOC 或 FP 的函数^①。LOC 或 FP 的值采用 24.6.3 节和 24.6.4 节所描述的方法进行估算，但不使用该节中的表，而是将 LOC 或 FP 的结果值代入到估算模型中。

关键点 估算模型反映的是导出其所基于的项目集，因此模型具有领域敏感性。

用以支持大多数估算模型的经验数据都是从有限的项目样本中得出的。因此，还没有一种估算模型能够适用于所有软件类型和开发环境。所以，从这些模型中得到的结果应该慎重使用。

应该对估算模型进行调整，以反映当前项目的情况。应该使用从已完成项目中收集的数据对该模型进行检验——方法是将数据代入到模型中，然后将实际结果与预测结果进行比较。如果两者一致性很差，则在使用该模型前，必须对其进行调整和再次检验。

24.7.1 估算模型的结构

典型的估算模型是通过将以往软件项目中收集的数据进行回归分析而导出的。这种模型的总体结构表现为下面的形式 [Mat94]：

$$E = A + B \times (e_v) C \quad (24.3)$$

其中， A 、 B 、 C 是经验常数， E 是工作量（以人月为单位）， e_v 是估算变量（LOC 或 FP）。除了式 (24.3) 所表示的关系外，大多数估算模型都有某种形式的项目调整成分，使得 E 能够根据其他的项目特性（例如，问题的复杂性、开发人员的经验、开发环境）加以调整。从任何一个根据经验得出的模型可以看出，必须根据项目特定环境的要求对估算模型进行调整。

24.7.2 COCOMO II 模型

Barry Boehm [Boe81] 在其关于“软件工程经济学”的经典著作中介绍了一种层次结构的软件估算模型，称为 COCOMO (COnstructive COst MOdel，构件性成本模型)。最初的 COCOMO 模型是得到产业界最广泛使用和讨论的软件成本估算模型之一。现在，它已经演化成更全面的估算模型，称为 COCOMOII [Boe00]。与其前身一样，COCOMOII 实际上也是一种层次结构的估算模型，应用于软件过程的不同“阶段”。

COCOMO II 模型与所有软件估算模型一样，也需要使用规模估算信息，在模型层次结构中有三种不同的规模估算选择：对象点^②、功能点和源代码行。

24.7.3 软件方程

软件方程 [Put92] 是一个动态的多变量模型，它假定在软件开发项目的整个生命周期中有特定的工作量分布。该模型是根据从 4000 多个当代软件项目中收集的生产率数据导出的。根据这些数据，我们导出以下形式的估算模型：

$$E = \frac{LOC \times B^{0.33}}{P^3} \times \frac{1}{t^4} \quad (24.4)$$

其中：

① 24.6.7 节给出了使用用例作为独立变量的经验模型。但是，到目前为止，在文献中出现的非常少。

② 对象点是一种间接的软件测量。计算对象点时，使用如下的计数值：(1) (用户界面的) 屏幕数；(2) 报表数；(3) 构造应用时可能需要的构件数，加上复杂度系数。

E 为工作量, 以人月或人年为单位。

t 为项目持续时间, 以月或年为单位。

B 为“特殊技能因子”^①。

P 为“生产率参数”, 它反映了: 总体的过程成熟度及管理实践; 采用良好的软件工程实践的程度; 使用的程序设计语言的水平; 软件环境的状态; 软件团队的技能和经验; 应用的复杂性。

对于实时嵌入式软件的开发, 典型值是 $P = 2000$; 对于电信及系统软件, $P = 10000$; 对于商业系统应用, $P = 28000$ 。当前情况下的生产率参数可以根据以往开发工作中收集到的历史数据来导出。

应该注意到, 软件方程有两个独立的参数: (1) 规模的估算值 (以 LOC 为单位); (2) 项目持续时间, 以月或年为单位。

Putnam 和 Myers[Put92] 为了简化估算过程, 并将估算模型表示成更通用的形式, 他们给出了一组从软件方程中导出来的方程式。最短开发时间定义为:

$$t_{\min} = 8.14 \frac{\text{LOC}}{P^{0.43}}, \text{ 以月为单位, 用于 } t_{\min} > 6 \text{ 个月的情况} \quad (24.5a)$$

$$E = 180Bt^3, \text{ 以人月为单位, 用于 } E \geq 20 \text{ 人月的情况} \quad (24.5b)$$

注意方程式 (24.5b) 中的 t 是以年为单位的。

对本章前面所讨论的 CAD 软件, 使用方程式 (24.5), 令 $P = 12000$ (对科学计算软件的推荐值):

$$t_{\min} = 8.14 \times \frac{33200}{12000^{0.43}} = 12.6 \text{ (月)}$$

$$E = 180 \times 0.28 \times 1.05^3 = 58 \text{ (人月)}$$

由软件方程得到的结果与 24.6 节产生的估算值非常一致。同 24.7.2 节中提到的 COCOMO 模型一样, 软件方程将继续演化下去, 关于该估算方法扩展版本的进一步讨论参见 [Put97b]。

24.8 面向对象项目的估算

使用明确为面向对象软件设计的估算技术来对软件成本估算的传统方法进行补充, 这种做法是值得的。Lorenz 和 Kidd[Lor94] 给出了下列方法:

1. 使用工作量分解、FP 分析和任何其他适用于传统应用的方法进行估算。
2. 使用需求模型 (第 9 章) 建立用例并确定用例数。要认识到随着项目的进展, 用例数可能会改变。
3. 由需求模型确定关键类 (在第 9 章中称为分析类) 的数量。
4. 对应用的界面类型进行归类, 以确定支持类的乘数。对应于没有图形用户界面、基于文本的用户界面、传统的图形用户界面、复杂的图形用户界面这 4 种界面类型, 相应的支持类乘数分别是 2.0、2.25、2.5 和 3.0。关键类的数量 (第 3 步) 乘上乘数就得到了支持类数量的估算值。
5. 将类的总数 (关键类 + 支持类) 乘以每个类的平均工作单元数。Lorenz 和 Kidd 建议

① 随着“对集成、测试、质量保证、文档和管理技能的需求的增长”, B 的值缓慢增加 [Put92]。对于较小的程序 ($\text{KLOC}=5 \sim 15$), $B=0.16$ 。对于超过 70KLOC 的程序, $B=0.39$ 。

每个类的平均工作单元数是 15 ~ 20 人日。

6. 将用例数乘以每个用例的平均工作单元数, 对基于类的估算做交叉检查。

习题与思考题

- 24.1 假设你是一家开发家用机器人软件公司的项目经理, 你已经承接了为草坪割草机器人开发软件的项目。写一个范围陈述来描述该软件, 确定你的范围陈述是“界定的”。如果你对机器人不熟悉, 在你开始写作之前先做一些调研工作。还要说明你对所需硬件的设想。或者, 你也可以选择其他感兴趣的问题, 而不做草坪割草机器人。
- 24.2 在 24.1 节简要讨论了软件项目的复杂性。列出影响项目复杂性的软件特性(例如, 并发操作、图形输出), 按其对项目的影响程度顺次排列。
- 24.3 在计划过程中, 性能是一个重要的考虑因素。针对不同的软件应用领域, 分别讨论如何以不同的方式来解释性能。
- 24.4 对你在问题 24.1 中描述的机器人软件进行功能分解。估算每个功能的规模(用 LOC)。假定你所在组织的平均生产率是 450LOC/pm, 劳动力价格是每人月 7000 美元, 使用本章所讲的基于 LOC 的估算技术来估算构建该软件所需的工作量及成本。
- 24.5 使用“软件方程”来估算草坪割草机器人软件。假设采用方程式(24.4), 且 $P=8000$ 。
- 24.6 建立一个电子表格模型, 实现本章所述的一种或多种估算技术。或者从基于 Web 的资源中获取一个或多个在线估算模型。
- 24.7 组建一个项目团队, 开发软件工具来实现本章所介绍的每种估算技术。
- 24.8 有一点似乎很奇怪: 成本和进度估算是在软件项目计划期间完成的——在详细的软件需求分析或设计之前进行。你认为为什么会这样? 是否存在不需要这样做的情况?

扩展阅读与信息资源

大多数软件项目管理书籍都包含了对项目估算的讨论。项目管理研究所(《PMBOK Guide》, PMI, 2001)、Wysoki(《Effective Project Management: Traditional, Agile, Extreme》, 6th ed., Wiley, 2011)、Lewis(《Project Planning Scheduling and Control》, 5th ed., McGraw-Hill, 2010)、Kerzner(《Project Management: A Systems Approach to Planning, Scheduling, and Controlling》, 10th ed., Wiley, 2009)、Bennatan(《On Time, Within Budget: Software Project Management Practices and Techniques》, 3rd ed., Wiley, 2000)以及Phillips[Phi98]都提供了有用的估算指南。

McConnell(《Software Estimation: Demystifying the Black Art》, Microsoft Press, 2006)撰写了实用指南, 为那些必须估算软件成本的人提供了有价值的指导。Parthasarathy(《Practical Software Estimation》, Addison-Wesley, 2007)强调以功能点作为估算度量。Hill(《Practical Software Project Estimation》, McGraw-Hill Osborne Media, 2010)以及Laird和Brennan(《Software Measurement and Estimation: A Practical Approach》, Wiley-IEEE Computer Society Press, 2006)论述了测量及其在软件估算中的应用。Pfleeger(《Software Cost Estimation and Sizing Methods, Issues and Guidelines》, RAND Corporation, 2005)给出了一个浓缩的指南, 描述了估算的很多基本原理。Jones(《Estimating Software Costs》, 2nd ed., McGraw-Hill, 2007)撰写的著作是对模型和数据最全面的论述之一, 这些模型和数据可用于各个应用领域的软件估算。Coombs(《IT Project Estimation》, Cambridge University Press, 2002)以及Roetzheim和Beasley(《Software Project Cost and Schedule Estimating: Best Practices》, Prentice Hall, 1997)介绍了很多有用的模型, 并逐步给出了生成最可能估算的指南。

大量的关于软件估算的信息可以在网上获得。最新的参考文献参见SEPA网站 www.mhhe.com/pressman 下的“software engineering resources”。

项目进度安排

要点浏览

概念: 你已经选择了合适的过程模型, 确定了必须完成的软件工程任务, 估算了工作量和人员数量, 明确了项目最后期限, 甚至已经考虑了风险, 现在是综合运用它们的时候了。也就是说, 你应该创建一个软件工程任务网络, 该网络将使你能够按时完成工作。任务网络创建完成之后, 你必须为每一个任务确定责任, 还要确保完成这些责任, 并在风险到来时调整该网络。简单地说, 这就是软件项目进度安排和跟踪。

人员: 在项目级, 是那些使用从软件工程师处获得的信息的软件项目管理者们。在个体级, 是软件工程师自己。

重要性: 为了建造复杂的系统, 很多软件工程任务会并行地进行, 而且在一个任务中得到的工作结果可能对在另一个任务中将要进行的工作具有深远的影响。

如果没有进度安排, 任务之间的这种相互依赖性将是非常难以理解的。实际上, 没有一个详细的进度安排, 要评估中等程度或大型的软件项目的进展情况也是不可能的。

步骤: 软件过程模型中规定的软件工程任务要根据具体实现的功能进行细化; 为每一个任务分配工作量和工期; 创建任务网络 (也称为“活动网络”), 使得软件团队能够在最后期限之前完成项目。

工作产品: 项目进度安排和相关的信息。

质量保证措施: 正确的进度安排要求: (1) 网络中包含所有的任务; (2) 给每个任务合理分配工作量和时间; (3) 明确指出任务间的依赖关系; (4) 资源应分配给具体要完成的工作; (5) 提供短时间间隔的里程碑, 以便于过程跟踪。

20 世纪 60 年代后期, 一位热情的青年工程师受命为一个自动制造业应用项目“编写”计算机程序。选择他的原因非常简单, 因为在整个技术小组中他是唯一参加过计算机编程培训班的人。这位工程师对汇编语言的 IN 和 OUT 指令以及 Fortran 语言有所了解, 但是却根本不懂软件工程, 更不用说项目进度安排和跟踪了。

他的老板给了他一大堆相关的手册, 口头描述了需要做些什么。年轻人被告知该项目必须在两个月之内完成。

他阅读了这些手册, 想好了解决方法, 就开始编写代码。两周之后, 老板将他叫到办公室询问项目进展情况。

“非常顺利,” 工程师以年轻人的热情回答道, “这个项目远比我想象的简单, 我差不多已经完成了 75% 的任务。”

关键概念

- 关键路径
- 挣值
- 工作量分配
- 人员与工作量
- 进度安排
- 原则
- 任务网络
- 时间盒
- 时序图
- 跟踪
- 工作分解

老板笑了，然后鼓励这个青年工程师继续努力工作，准备好一周后再汇报工作进度。

一周之后老板将年轻人叫到办公室，问道：“现在进度如何？”

“一切顺利，”年轻人回答说，“但是我遇到了一些小麻烦。我会排除这些困难，很快就可以回到正轨上来。”

“你觉得在最后期限之前能完成吗？”老板问道。

“没问题，”工程师答道，“我差不多已经完成90%了。”

如果你在软件领域中工作过几年，你一定可以将这个故事写完。毫不奇怪，青年工程师在整个项目工期内始终停留在90%的进度上，实际上直到交付期限之后一个月（在别人的帮助下）才完成。^①

在过去的50年间，这样的故事在不同的软件开发者中已经重复了成千上万次，这是为什么呢？

25.1 基本概念

虽然软件延期交付的原因很多，但是大多数都可以追溯到下面列出的一个或多个根本原因上：

- 不切实际的项目最后期限，由软件团队以外的某个人制定，并强加给软件团队的管理者和开发者。
- 客户需求发生变更，而这种变更没有在项目变更进度表上预先安排。
- 对完成该工作所需的工作量和资源数量估计不足。
- 在项目开始时，没有考虑到可预测的和不可预测的风险。
- 出现了事先无法预计的技术难题。
- 出现了事先无法预计的人力问题。
- 由于项目团队成员之间的交流不畅而导致的延期。
- 项目管理者未能发现项目进度拖后，也未能采取措施来解决这一问题。

在软件行业中，人们对过于乐观的（即“不切实际的”）项目最后期限已经司空见惯。从设定项目最后期限的人的角度来看，有时候这样的项目最后期限是合理的。但是常识告诉我们，合理与否还必须由完成工作的人员来判断。

第24章讨论的估算方法和本章中的进度安排技术，通常都需要在规定的项目最后期限约束下进行。如果最乐观的估算都表明该项目最后期限是不现实的，一个称职的项目管理者就应该“保护其团队免受不适当的（进度安排）压力……（并）将这种压力反映给施加压力的一方”[Pag85]。

举例说明，假定你负责的软件团队受命开发一个医疗诊断仪器的实时控制器，该控制器要在9个月之内推向市场。在你进行了仔细的估算和风险分析（第26章）之后得到的结论是：在现有人员条件下，需要14个月的时间才能完成这一软件。你下一步该怎么办呢？

闯进客户的办公室（这里的客户很可能是市场/营销人员）并要求修改交付日期似乎不太现实。外部市场压力已经决定了交付日期，届时必须发布产品。而（从事业前途的角度出

引述 在所有的软件项目中，过于理性或缺少理性的进度安排可能最具破坏性影响。

Capers Jones

引述 我热爱最后期限，我喜欢它们飞过时发出的声音。

Douglas Adams

^① 你可能觉得惊奇，但这个故事是我自己经历过的事（RSP）。

发) 拒绝这一项目同样是鲁莽的。那么应该怎么办呢? 在这种情况下, 建议按照以下步骤进行处理:

1. 按照以往项目的历史数据进行详细的估算, 确定项目的估算工作量和工期。
2. 采用增量过程模型(第4章)制定软件工程策略, 以保证能够在规定的交付日期提供主要功能, 而将其他功能的实现推到以后。然后将这一计划做成文档。
3. 与客户交流, 并(用详细的估算结果)说明为什么规定的交付日期是不现实的。一定要指出所有这些估算都是基于以往的项目实践, 而且为了在目前规定的交付期限内完成该项目, 与以往相比在工作效率上必须提高的百分比^①。可以做如下解释:

我认为在 XYZ 控制器软件的交付日期方面存在问题, 我已经将一份以往项目中生产率的简化明细分类表和以多种不同方式进行的项目估算提交给各位, 你们会注意到, 在假设与以往的生产率相比有 20% 的提高的情况下, 交付时间仍然需要 14 个月而不是 9 个月。

4. 将增量开发策略作为可选计划提交给客户:

我们有几种方案, 我希望各位能够在这些方案中做出决策。第一个方案, 我们可以增加预算, 并引入额外的资源, 以使我们能够在 9 个月内完成这项工作。但是应该知道由于时间限制过于苛刻, 这样做将会增加质量变差的风险^②; 第二个方案, 去掉一部分需求中所列出的软件功能和特性, 由此得到功能稍弱的产品的最初版本, 但是我们可以对外宣布全部功能, 并在总共 14 个月的时间内交付这些功能; 第三个方案, 是不顾现实条件的约束, 而希望项目能够在 9 个月时间内完成, 结果是我们竭尽全力, 但是却无法向客户提供任何功能。我希望你们会赞成我的观点, 第三个方案是不可行的。过去的历史和我们最乐观的估算都表明这是不现实的, 是在制造一场灾难。

尽管这样做会有人抱怨, 但如果你给出了基于准确历史数据的可靠估算, 那么最终的谈判结果将可能是选择方案 1 或方案 2。不现实的交付期限就不存在了。

25.2 项目进度安排概述

曾经有人向 Fred Brooks 请教软件项目的进度是怎样被延误的? 他的回答既简单又深刻: “某天某时。”

技术性项目(不论它是涉及水力发电厂建设, 还是操作系统开发)的现实情况是: 在实现大目标之前必须完成数以百计的小任务。这些任务中有些是处于主流之外的, 其进度不会影响到整个项目的完成日期。而有些任务却是位于“关键路径”上, 如果这些“关键”任务的进度拖后, 则整个项目的完成日期就会受到威胁。

项目管理者的职责是确定所有的项目任务, 建立相应的网络来描述它们之间的依赖关系, 明确网络中的关键任务, 然后跟踪关键任务的进展, 以确保能够在“某天某时”发现进度延误情况。为了做到这一点, 管理者

提问 当管理者要求的项目最后期限无法实现时, 应该怎么办?

建议 为达到项目管理者的目标所必须完成的任务不应该以手工方式来安排, 有很多优秀的项目进度安排工具可供使用。

① 如果生产率提高 10% ~ 25%, 实际上是有可能完成这个项目的。但大多数情况是, 所需提高的团队生产率要高于 50%。所以说这是不切实际的期望。

② 你还可以补充说: “人员数量的增加不会成比例地缩短完成该项目所需要的时间。”

必须建立相当详细的进度表,使得项目管理者能够监督进度,并控制整个项目。

软件项目进度安排 (software project scheduling) 是一种活动,它通过将工作量分配给特定的软件工程任务,从而将所估算的工作量分配到计划的项目工期内。但要注意的是,进度是随时间而不断演化的。在项目计划早期,建立的是一张宏观进度表,该进度表标识了所有主要的过程框架活动和这些活动所影响的产品功能。随着项目的进展,宏观进度表中的每个条目都会被细化成详细的进度表,这样就标识了特定的(完成一个活动所必须实现的)软件活动和任务,同时也进行了进度安排。

可以从两个不同的角度来讨论软件工程项目的进度安排。第一种情况,计算机系统的最终发布日期已经确定(而且不能更改),软件开发组织必须将工作量分布在预先确定的时间框架内。第二种情况,假定已知大致的时间界限,但是最终发布日期由软件工程开发组织自行确定,工作量是以能够最好地利用资源的方式来进行分配,而且在对软件进行仔细分析之后才决定最终发布日期。但不幸的是,第一种情况发生的频率远远高于第二种情况。

引述 过于乐观的进度安排并不会缩短实际进度,反而会拖后进度。
Steve McConnell

25.2.1 基本原则

就像软件工程的所有其他领域一样,软件项目进度安排也有很多的基本指导原则:

划分 (compartmentalization)。必须将项目划分成多个可以管理的活动和任务。为了实现项目的划分,产品和过程都需要进行分解。

相互依赖性 (interdependency)。划分后的各个活动或任务之间的相互依赖关系必须是明确的。有些任务必须按顺序出现,而有些任务则可以并发进行。有些活动只有在其他活动产生的工作产品完成后才能够开始,而有些则可以独立进行。

时间分配 (time allocation)。每个要进行进度安排的任务必须分配一定数量的工作单位(例如,若干人日的工作量)。此外,还必须为每个任务指定开始日期和完成日期,任务的开始日期和完成日期取决于任务之间的相互依赖性以及工作方式是全职还是兼职。

工作量确认 (effort validation)。每个项目都有预定人员数量的软件团队参与。在进行时间分配时,项目管理者必须确保在任意时段中分配的人员数量不会超过项目团队中的总人员数量。例如,某项目分配了 3 名软件工程师(例如,每天可分配的工作量为 3 人日[⊖])。在某一天中,需要完成 7 项并发的任务,每个任务需要 0.5 人日的工作量,在这种情况下,所分配的工作量就大于可供分配的工作量。

确定责任 (defined responsibility)。进度计划安排的每个任务都应该指定特定的团队成员来负责。

明确输出结果 (defined outcome)。进度计划安排的每个任务都应该有一个明确的输出结果。对于软件项目而言,输出结果通常是一个工作产品(例如,一个模块的设计)或某个工作产品的一部分。通常可将多个工作产品组合成可交付产品。

确定里程碑 (defined milestone)。每个任务或任务组都应该与一个项目里程碑相关联。当

关键点 在进度安排时,要划分工作,描述任务间的相互依赖性,为每个任务分配工作量和时间,确定责任、输出结果和里程碑。

[⊖] 实际上,由于与工作无关的会议、病假、休假以及各种其他原因,可供分配的工作量要少于 3 人日。但在这里,我们假定员工时间是 100% 可用的。

一个或多个工作产品经过质量评审（第 15 章）并且得到认可时，就标志着一个里程碑的完成。随着项目进度的推进，会应用到上述的每一条原则。

25.2.2 人员与工作量之间的关系

许多负责软件开发工作的管理者仍然普遍坚信这样一个神话：“即使进度拖后，我们也总是可以在项目后期增加更多的程序员来跟上进度。”不幸的是，在项目后期增加人手通常会对项目产生破坏性的影响，其结果是使进度进一步拖延。后期增加的人员必须学习这一系统，而培训他们的人员正是那些一直在工作着的人，当他们进行教学时，就不能完成任何工作，从而使项目进一步延后。

建议 如果必须给一个已经拖延的项目增加人员，就要确保已将详细划分的任务分配给他们。

除去学习系统所需的时间之外，整个项目中，新加入人员将会增加人员之间交流的路径数量和交流的复杂度。虽然交流对于一个成功的软件开发项目而言绝对是必不可少的，但是每增加一条新的交流路径就会增加额外的工作量，从而需要更多的时间。

多年以来的经验数据和理论分析都表明项目进度是具有弹性的。即在一定程度上可以缩短项目交付日期（通过增加额外资源），也可以拖延项目交付日期（通过减少资源数量）。

PNR（Putnam-Norden-Rayleigh）曲线^①表明了一个软件项目中所投入的工作量与交付时间的关系。项目工作量和交付时间的函数关系曲线如图 25-1 所示。图中的 t_0 表示项目最低交付成本所需的最少时间（即花费工作量最少的项目交付时间），而 t_0 左边（即当我们想提前交付时）的曲线是非线性上升的。

关键点 PNR 曲线表明，拖延项目交付时间可以显著地降低项目成本。

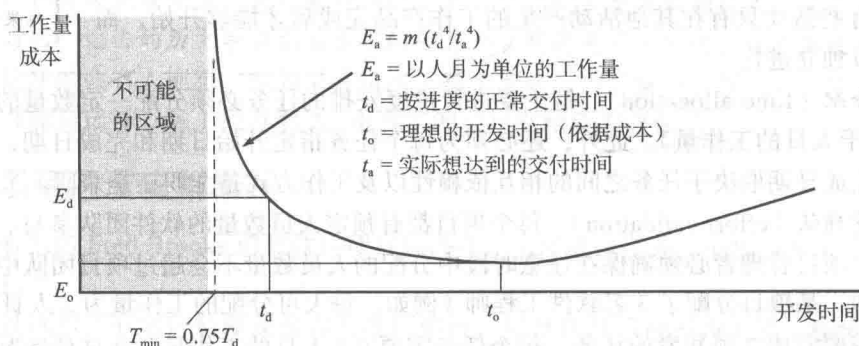


图 25-1 工作量和交付时间的关系

举一个例子，假设一个软件项目团队根据进度安排和现有的人员配置，估算所需要的工作量应为 E_d ，正常的交付时间应为 t_d 。虽然可以提前交付，但曲线在 t_d 的左侧急剧上升。事实上，PNR 曲线不仅说明了项目的交付时间不能少于 $0.75t_d$ ，如果想更少，项目会进入“不可能的区域”，并面临着很高的失败风险；还说明了最低成本的交付时间 t_0 应该满足 $t_0 = 2t_d$ ，即拖延项目交付可以明显降低成本，当然，这里的成本必须将与延期相关的营销成本排除在外。

在第 24 章中介绍的软件方程 [Put92] 就是来源于 PNR 曲线，它表明了完成一个项目的

① 相关的初始研究参见 [Nor70] 和 [Put78]。

时间与投入该项目的人员工作量之间是高度非线性的关系。交付的代码（源程序代码）行数 L 与工作量和开发时间的关系可以用下面的公式表示：

$$L = P \times E^{1/3} t^{4/3}$$

其中， E 是以人月为单位的开发工作量； P 是生产率参数，它反映了影响高质量软件工程工作的各种因素的综合效果（ P 通常在 2000 到 12000 之间取值）； t 是以月为单位的项目工期。

重新调整这个软件方程，可以得到开发工作量 E 的计算公式：

$$E = \frac{L^3}{P^3 t^4} \quad (25.1)$$

建议 离项目的交付日期越来越近时，你意识到，不管参与工作的人数是多少，工作均不能按计划完成。那就面对现实，确定新的交付日期。

其中， E 是在软件开发和维护的整个生命周期内所需的工作量（按人年计算）； t 是以年为单位的开发时间；引入平均劳动力价格因素（\$/人年）之后，开发工作量的计算公式还能够与开发成本相关联。

这一方程式引出了一些有趣的结果。假设有一个复杂的实时软件项目，估计需要 33000 源代码行和 12 人年的工作量。如果项目团队有 8 个人，那么项目大约需要 1.3 年的时间完成。但是如果将交付日期延长到 1.75 年，则由式（25.1）所描述的模型所具有的高度非线性特性将得出以下结论：

$$E = \frac{L^3}{P^3 t^4} \approx 3.8 \text{ 人年}$$

这意味着通过将交付日期推迟 6 个月，我们可以将项目团队的人数从 8 人减少到 4 人！这一结果的有效性有待考证，但是其含意却十分清楚：通过在略为延长的时间内使用较少的人员，可以实现同样的目标。

25.2.3 工作量分配

在第 24 章中讨论的各种软件项目估算技术最终都归结为对完成软件开发所需工作单位（如人月）的估算。软件过程中的工作量分配通常采用 40-20-40 法则。总体工作量的 40% 分配给前期的分析和设计，40% 用于后期测试。因此，你可以推断出编码工作（20% 的工作量）是次要的。

提问 在软件过程的工作流程中应如何分配工作量？

这种工作量分配方法只能作为指导原则^①。各个项目的特点决定了其工作量如何分配。用于项目计划的工作量很少超过 2% ~ 3%，除非提交给组织的项目计划费用极高而且具有高风险。客户交流与需求分析大约占用 10% ~ 25% 的项目工作量，用于分析或原型开发的工作量应该与项目规模和复杂度成正比地增长。通常有 20% ~ 25% 的工作量用于软件设计，用于设计评审和随之而来的迭代开发也必须考虑。

因为在软件设计时投入了相当的工作量，所以随后的编码工作就变得相对简单。总体工作量的 15% ~ 20% 就可以完成这一工作。测试和随之而来的调试工作将占用 30% ~ 40% 的软件开发工作量。软件的重要性决定了所需测试工作的分量，如果软件系统是人命关天的（即软件错误可能使人丧命），就应该考虑分配更高的测试工作量比例。

① 现在，40-20-40 法则不再适用。有些人认为用于分析和设计的工作量应超过总工作量的 40%。而相反的是，敏捷开发的倡导者（第 5 章）认为“前期”工作应该越快越好，团队应该快速进入构建阶段。

25.3 为软件项目定义任务集

无论选择哪一种过程模型，一个软件团队要完成的工作都是由任务集组成的，这些任务集使得软件团队能够定义、开发和最终维护计算机软件。没有能普遍适用于所有软件项目的任务集。适用于大型复杂系统的任务集可能对于相对简单的小型软件项目而言就过于复杂。因此，有效的软件过程应该定义一组任务集来满足不同类型项目的要求。

同第3章介绍的一样，任务集中包含了为完成某个特定项目所必须完成的所有软件工作任务、里程碑、工作产品以及质量保证过滤器。为了获得高质量的软件产品，任务集必须提供充分的规程要求，但同时又不能让项目团队负担不必要的工作。

在进行项目进度安排时，必须将任务集分布在项目时序图上。任务集应该根据软件团队所决定的项目类型和严格程度而有所不同。尽管很难建立一个全面详尽的软件项目分类方法，但是大多数软件组织遇到的项目一般属于下述类型：

1. 概念开发项目（concept development project）。目的是为了探索某些新的业务概念或者某种新技术的应用。
2. 新应用开发（new application development）项目。根据特定的客户需求而承担的项目。
3. 应用增强（application enhancement）项目。对现有软件中最终用户可见的功能、性能或界面进行修改。
4. 应用维护项目（application maintenance project）。以一种最终用户不会立即察觉到的方式对现有软件进行纠错、修改或者扩展。
5. 再工程项目（reengineering project）。为了全部或部分重建一个现有（遗留）系统而承担的项目。

即使在单一的项目类型中，也会有许多因素影响任务集的选择。[Pre05]中描述了很多因素：项目的规模、潜在的用户数量、任务的关键性、应用程序的寿命、需求的稳定性、客户/开发者进行沟通的容易程度、可应用技术的成熟度、性能约束、嵌入式和非嵌入式特性、项目人员配置以及再工程因素等。综合考虑这些因素就形成了严格程度（degree of rigor）的指标，它将应用于所采用的软件过程中。

网络资源 适应性过程模型（Adaptable Process Model, APM）可以辅助不同的软件项目定义各自的任务集。有关APM的详细信息见 www.rspa.com/apm。

25.3.1 任务集举例

概念开发项目是在探索某些新技术是否可行时发起的。这种技术是否可行尚不可知，但是某个客户（如营销人员）相信其具有潜在的利益。概念开发项目的完成需要应用以下主要任务：

- 1.1 确定概念范围。确定项目的整体范围。
- 1.2 初步的概念策划。确定为承担项目范围所涵盖的工作组织应具有的工作能力。
- 1.3 技术风险评估。评估与项目范围中将要实现的技术相关联的风险。
- 1.4 概念证明。证明新技术在软件环境中的生命力。
- 1.5 概念实现。以可以由客户方进行评审的方式实现概念的表示，而且在将概念推荐给其他客户或管理者时能够用于“营销”目的。
- 1.6 客户反应。向客户索取对新技术概念的反馈，并以特定的客户应用作为目标。

快速浏览以上任务，你应该不会有任何诧异。实际上概念开发项目的软件流程（以及其他所有类型的项目）与人们的常识相差无几。

25.3.2 主要任务的细化

上一节中所描述的主要任务（如软件工程活动）可以用来制定项目的宏观进度表。但是，必须将宏观进度表进行细化，以创建详细的项目进度表。细化工作始于将每项主要任务分解为一组子任务（以及相关的工作产品和里程碑）。

这里以“任务1.1 确定概念范围”的任务分解为例。任务细化可以使用大纲格式，但是在这里将使用过程设计语言来说明“确定概念范围”这一活动的流程。

任务定义：任务1.1 确定概念范围

1.1.1 确定需求、效益和潜在的客户

1.1.2 确定所希望的输出 / 控制和驱动应用程序的输入事件

开始任务1.1.2

1.1.2.1 TR：评审需求的书面描述[⊖]

1.1.2.2 导出客户可见的输出 / 输入列表

1.1.2.3 TR：与客户一起评审输出 / 输入，并在需要时进行修改

结束任务1.1.2

1.1.3 为每个主要功能定义功能 / 行为

开始任务1.1.3

1.1.3.1 TR：评审在任务1.1.2中得到的输出和输入数据对象

1.1.3.2 导出功能 / 行为模型

1.1.3.3 TR：与客户一起评审功能 / 行为模型，并在需要时进行修改

结束任务1.1.3

1.1.4 把需要在软件中实现的技术要素分离出来

1.1.5 研究现有软件的可用性

1.1.6 确定技术可行性

1.1.7 对系统规模进行快速估算

1.1.8 创建“范围定义”

结束任务1.1的任务定义

在过程设计语言中标注的这些任务和子任务共同构成了“确定概念范围”这个行动的详细进度表的基础。

25.4 定义任务网络

单个任务和子任务之间在顺序上存在相互依赖的关系。而且，当有多人参与软件工程项目时，多个开发活动和任务并行进行的可能性很大。在这种情况下，必须协调多个并发任务，以保证它们能够在后继任务需要其工作产品之前完成。

任务网络（task network）也称为活动网络（activity network），是项目任务流程的图形表示。有时将任务网络作为向自动项目进度安排工具中输

关键点 任务网络是描述任务间依赖关系和确定关键路径的有效机制。

⊖ TR 表示在此需要进行一次技术评审。

入任务序列和依赖关系的机制。最简单的任务网络形式（创建宏观进度表时使用）只描述了主要的软件工程任务。概念开发项目的任务网络示意图如图 25-2 所示。

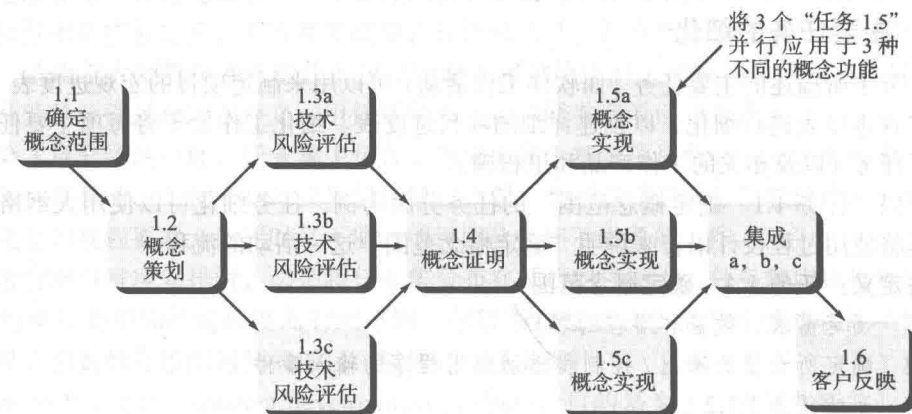


图 25-2 概念开发项目的任务网络

软件工程活动的并发本质导致了在进度安排上有很多要求。由于并行任务是异步发生的，所以项目管理者必须确定任务之间的依赖关系，以保证项目朝着最终完成的方向持续发展。另外，项目管理者应该注意那些位于关键路径（critical path）上的任务。也就是说，为了保证整个项目如期完成，必须保证这些任务能够如期完成。在本章的后面将详细讨论这些问题。

值得注意的是，图 25-2 中所示的任务网络是宏观的。详细的任务网络（详细进度表的前身）中应该对图 25-2 所示的各个活动加以扩展。例如，应该扩展任务 1.1，以表现 34.3.2 节所述的任务 1.1 细化中的所有任务。

25.5 进度安排

软件项目的进度安排与任何其他多任务工程工作的进度安排几乎没有差别。因此，通用的项目进度安排工具和技术不必做太多修改就可以应用于软件项目。

进度计划评估及评审技术（Program Evaluation and Review Technique, PERT）和关键路径方法（Critical Path Method, CPM）就是两种可以用于软件开发的项目进度安排方法。这两种技术都是由早期项目计划活动中已经产生的信息来驱动的，早期的项目计划活动包括：工作量的估算、产品功能的分解、适当过程模型和任务集的选择，以及所选择的任务的分解。

任务之间的依赖关系可以通过任务网络来确定。任务有时也称为项目的工作分解结构（Work Breakdown Structure, WBS），可以是针对整个产品，也可以是针对单个功能来进行定义。

PERT 和 CPM 两种方法都是定量划分的工具，可以使软件计划者完成：（1）确定关键路径——决定项目工期的任务链；（2）基于统计模型为单个任务进行“最有可能”的时间估算；（3）为特定任务的时间“窗口”计算“边界时间”。

引述 唯一要我们做出决定的就是怎样分配所给定的时间。

Gandalf,《指环王：护戒使者》

软件工具 项目进度安排

[目标] 项目进度安排工具的目标是使项目管理者能够确定工作任务，建立工作任务之间的依赖关系，为工作任务分配人员，并能够形成各种图表，以辅助对软件项目进行跟踪和控制。

[机制] 通常，项目进度安排工具要求完成各任务的工作分解结构的规格说明，或者创建任务网络。一旦确定了工作分解结构（大纲形式）或任务网络，就可以为每一个任务指定开始和结束日期、分配人员、确定交付日期以及其他内容。然后，项目进度安排工具可以生成多种时序图及其他表格，使项目管理者能够对项目的任务流程进行评估。随着项目的进展，这些信息可

以不断地进行更新。

[代表性工具]^①

- AMS Realtime。由 Advanced Management Systems (www.amsusa.com) 开发，可以对各种规模和类型的项目做进度安排。
- Microsoft Project。由 Microsoft (www.microsoft.com) 开发，是一个使用最广泛的 PC 项目进度安排工具。
- 4C。由 4C Systems (www.4csys.com) 开发，支持项目计划的各个方面，包括进度安排。

项目管理软件厂商及其产品的详细清单见 www.infogoal.com/pmc/pmcswr.htm。

25.5.1 时序图

在创建软件项目进度表时，计划者可以从一组任务（工作分解结构）入手。如果使用自动工具，就可以采用任务网络或者任务大纲的形式输入工作分解结构，然后再为每一项任务输入工作量、工期和开始日期。此外，还可以将某些任务分配给特定的人员。

输入信息之后，就可以生成时序图（timeline chart），也叫作甘特图（Gantt chart）。可以为整个项目建立一个时序图，也可以为各个项目功能或各个项目参与者分别建立各自的时序图。

图 25-3 给出了时序图的格式，该图描述了字处理（Word-Processing, WP）软件产品中确定概念范围这一任务的软件项目进度安排。所有的项目任务（针对“确定概念范围”）都在左边栏中列出。水平条表示各个任务的工期，当同一时段中存在多个水平条时，就代表任务之间的并发性，菱形表示里程碑。

输入了生成时序图所需的信息之后，大多数软件项目进度安排工具都能生成项目表（project table）——列出所有项目任务的表格，项目表中列出了各个任务计划的开始与结束日期、实际开始日期与结束日期以及各种相关信息（图 25-4）。通过项目表与时序图，项目管理者就可以跟踪项目的进展情况。

关键点 通过时序图可以确定在特定时间点将进行什么任务。

25.5.2 跟踪进度

如果制定正确，项目进度表应该成为一个能够确定在项目进展过程中跟踪和控制任务及里程碑的线路图。项目跟踪可以通过以下方式实现：

引述 软件状态报告的基本规则可以归纳为一句话：一点都不奇怪。
Capers Jones

^① 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

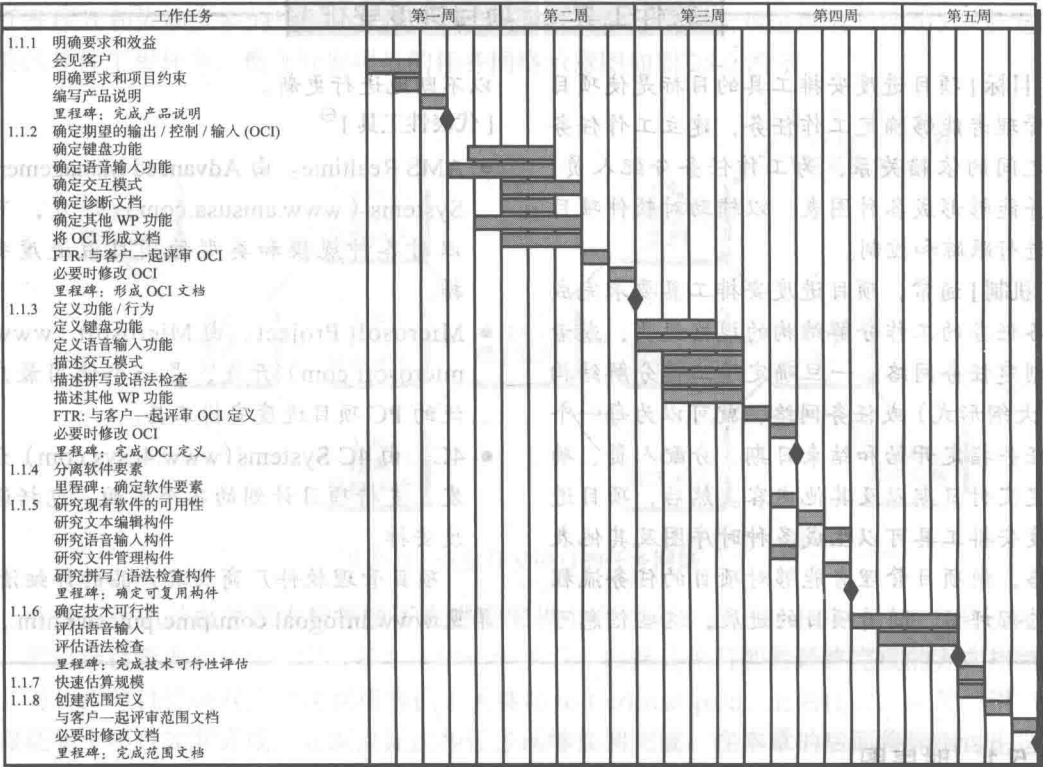


图 25-3 一个时序图的例子

工作任务	计划开始	实际开始	计划完成	实际完成	人员分配	工作量分配	备注
1.1.1 明确要求和效益							
会见客户	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
明确要求和项目约束	wk1, d2	wk1, d2	wk1, d3	wk1, d3	JPP	1 p-d	
编写产品说明	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
里程碑：完成产品说明							
1.1.2 确定期望的输出/控制/输入(OCI)							
确定键盘功能	wk1, d4	wk1, d4	wk2, d2	wk2, d2	BLS	1.5 p-d	
确定语音输入功能	wk1, d3	wk1, d3	wk2, d2	wk2, d2	JPP	2 p-d	
确定交互模式	wk2, d1		wk2, d3	wk2, d3	MLL	1 p-d	
确定诊断文档	wk2, d1		wk2, d2	wk2, d2	BLS	1.5 p-d	
确定其他 WP 功能	wk1, d4	wk1, d4	wk2, d3	wk2, d3	JPP	2 p-d	
将 OCI 形成文档	wk2, d1		wk2, d3	wk2, d3	MLL	3 p-d	
FTR: 与客户一起评审 OCI	wk2, d3		wk2, d3	wk2, d3	all	3 p-d	
必要时修改 OCI	wk2, d4		wk2, d4	wk2, d4	all	3 p-d	
里程碑：形成 OCI 文档	wk2, d5		wk2, d5	wk2, d5			
1.1.3 定义功能/行为							

图 25-4 一个项目表的例子

- 定期举行项目状态会议，由项目团队中的各成员分别报告进度和存在的问题。
- 评估在软件工程过程中所进行的所有评审的结果。
- 判断正式的项目里程碑（图 25-3 中的菱形）是否在预定日期内完成。
- 比较项目表（图 25-4）中列出的各项任务的实际开始日期与计划开始日期。

- 与开发人员进行非正式会谈，获取他们对项目进展及可能出现的问题的客观评估。
- 通过挣值分析（25.6 节）来定量地评估项目进展。

实际上，有经验的项目管理者会使用所有这些跟踪技术。

软件项目管理者通过施加控制来管理项目资源、处理问题和指导项目参与者。如果一切顺利（即项目在预算范围内按进度进行，评审结果表明的确取得了实际进展，达到了各个里程碑），则几乎不必施加控制。但是如果出现问题，项目管理者就必须施加控制，以便尽快解决问题。当诊断出问题之后，可能需要增加额外的资源来解决问题：雇用新员工或者重新安排项目进度。

建议 项目进展的最佳指标就是所定义的软件工作产品的实现和成功评审。

在面对交付期限的巨大压力时，有经验的项目管理者有时会使用一种称为时间盒（time-boxing）[Jal04] 的项目进度安排与控制技术。时间盒方法认为完整的产品可能难以在预定时间内交付，因此，应该选择增量软件开发范型（第 4 章），并为每个增量的交付制定各自的进度表。

接着，对与每个增量相关的任务实行时间盒技术。也就是按该增量的交付日期向后进行推算来调整各个任务的进度。将各个任务放入相应的“盒子”中，当一个任务触及其时间盒边界时（ $\pm 10\%$ 的范围内），则该项任务停止，下一任务开始。

对时间盒方法的最初反应通常是消极的：“如果工作尚未完成，我们该如何继续？”这个问题的答案在于完成工作的方式。当遇到时间盒的边界时，很可能已经完成了任务的 90%^①，余下 10% 的工作尽管重要，但是可以：（1）推迟到下一个增量中；或（2）在以后需要时再完成。项目朝着交付日期推进，而不是“卡”在某项任务上。

25.5.3 跟踪面向对象项目的进展

虽然迭代模型是最好的面向对象项目框架，但是，任务的并行性使得面向对象项目很难跟踪。困难在于项目管理者很难为面向对象项目建立有意义的里程碑，因为很多不同事物都是同时发生的。通常，有相应的准则来衡量下列主要的里程碑是否已经“完成”。

技术里程碑：面向对象分析完成

- 已经定义和评审了所有的类和类层次。
- 已经定义和评审了与每一个类相关的属性和操作。
- 已经建立和评审了各类之间的关系（第 9 章）。
- 已经建立和评审了行为模型（第 10 章）。
- 已经确定了可复用的类。

技术里程碑：面向对象设计完成

- 已经确定和评审了子系统集合。
- 各类已经分配给相应的子系统，并且已经通过评审。
- 已经建立和评审了任务分配。
- 已经明确责任和协作。
- 已经设计和评审了属性和操作。

① 爱冷嘲热讽的人也许会想起一句谚语：“完成系统的前 90% 需要 90% 的时间，完成剩下的 10% 也要用 90% 的时间。”

- 已经创建和评审了通信模型。

技术里程碑：面向对象程序设计完成

- 按照设计模型，每一个新类都已经编码实现。
- (从可复用库中) 提取的类已经实现。
- 已经构建了原型或增量。

技术里程碑：面向对象测试

- 已经评审了面向对象分析和设计模型的正确性和完整性。
- 已经建立和评审了类-职责-协作者网络(第9章)。
- 已经设计了测试用例，并且已经对每个类进行了类级测试(第19章)。
- 已经设计了测试用例，并且已经完成簇测试(第19章)，已经完成类的集成。
- 已经完成系统级测试。

建议 调试和测试是相辅相成的，通常可以通过考察“公开的”错误(缺陷)类型和数量来判断测试的状态。

就像前面介绍的，建立面向对象过程模型是以迭代方式进行的，在交付不同的增量给用户时，上述的每一个里程碑都可以进行修订。

25.6 挣值分析

在34.5节，我们讨论了一系列项目跟踪的定性方法，为项目管理者提供了项目进展情况的指标。但是，对所提供信息的评估在某种程度上是主观的。那么当软件团队按项目进度表实施工作任务时，是否存在某种量化的技术来评估项目进展情况呢？事实上，确实存在一种用于项目进展的定量分析技术，称为挣值分析(Earned Value Analysis, EVA)。Humphrey[Hum95]对挣值给出了如下讨论：

关键点 挣值提供了定量的项目进展指标。

不管要完成何种类型的工作，挣值系统为每个(软件项目)任务提供了通用的值尺度，可以估算完成整个项目所需要的总小时数，并且可以根据各个任务所估算的小时数占总小时数的百分比来确定该任务的挣值。

更简单地说，挣值是对项目进展的测量。它使得计划者能够不依赖于感觉，而是采用定量的分析方法来评估一个项目的“完成百分比”。事实上，Fleming和Koppleman[Fle98]就认为挣值分析“早在项目进展的前15%就提供了精确而可靠的性能数据”。按照以下步骤可以确定挣值。

1. 为进度表中的每个工作任务确定其预计工作的预算成本(Budgeted Cost of Work Scheduled, BCWS)。在估算过程中，要计划每个软件工程任务的工作量(以人时或人日为单位)，因此， $BCWS_i$ 是指工作任务*i*的计划工作量。为了确定在项目进度表中某特定时间点的项目进展状况，BCWS的值是在项目进度表中该时间点应该完成的所有工作任务的 $BCWS_i$ 值之和。
2. 所有工作任务的BCWS值加起来，可计算出完成工作的预算(Budget at Completion, BAC)，因此，对所有任务*k*，有

提问 如何计算挣值以评估项目进展。

$$BAC = \sum(BCWS_k)$$

3. 接着, 计算已完成工作的预算成本 (Budgeted Cost of Work Performed, BCWP)。BCWP 的值是在项目进度表中该时间点已经实际完成的所有工作任务的 BCWS 值之和。

Wilkens[Wil99] 指出: “BCWS 和 BCWP 的不同点是, 前者表示计划将完成的工作的预算, 后者表示已实际完成的工作的预算。” 给定 BCWS、BAC 和 BCWP 的值, 就可以得出相关的项目进展指标:

$$\text{进度表执行指标 SPI} = \frac{\text{BCWP}}{\text{BCWS}}$$

$$\text{进度表偏差 SV} = \text{BCWP} - \text{BCWS}$$

其中, SPI 是效率指标, 指出项目使用预定资源的效率, SPI 值越接近 1.0 说明项目的执行效率越高。SV 只表示与计划进度的偏差。

$$\text{预定完成百分比} = \frac{\text{BCWS}}{\text{BAC}}$$

表示在时间点 t 应该完成工作的百分比值。

$$\text{完成百分比} = \frac{\text{BCWS}}{\text{BAC}}$$

表示在特定时间点 t 实际完成工作的百分比值。

也可以计算出已完成工作的实际成本 (Actual Cost of Work Performed, ACWP)。ACWP 的值是在项目进度表中某时间点已经完成的工作任务的实际工作量之和。然后, 再计算:

$$\text{成本执行指标 CPI} = \frac{\text{BCWP}}{\text{ACWP}}$$

$$\text{成本偏差 CV} = \text{BCWP} - \text{ACWP}$$

CPI 值越接近 1.0 说明项目与预算越接近。CV 表示在项目特定阶段的成本节省 (相对于计划成本) 或短缺。

就像超视距雷达一样, 挣值分析在可能出现问题之前就指出了进度安排的难点, 这使得软件项目管理者能够在项目危机出现前采取有效措施。

习题与思考题

- 25.1 “不合理的”项目最后期限是软件行业中存在的现实情况。当你遇到这种情况时应该如何处理?
- 25.2 宏观进度表和详细进度表的区别是什么? 是否有可能只依据所制定的宏观进度表来管理一个项目? 为什么?
- 25.3 是否存在这种情况: 一个软件项目里程碑没有与某个评审相关联? 如果有, 请至少给出一个例子。
- 25.4 当多个人员参与软件项目时, 就有可能产生“交流开销”。与其他人员进行交流要花费时间, 这样就会降低个人生产率 (LOC/月), 最终导致整个团队生产率下降。(举几个例子) 量化说明非常精通软件工程实践和运用技术评审的软件工程师是如何提高团队生产率的 (与个人生产率的总和进行比较)。提示: 假设评审减少了返工, 而返工可能占一个人 20% ~ 40% 的时间。
- 25.5 尽管为延迟的软件项目增加人手可能会进一步拖延工期, 但是否在某些情况下并非如此呢? 请说明。
- 25.6 人员和时间的关系是高度非线性的。使用 Putnam 的软件方程 (25.2.2 节) 编制一个表, 以反映软件项目中人员数量与项目工期之间的关系——该项目需要 50000 LOC 和 15 人年的工作量 (生

网络资源 有关挣值分析的信息源见 <http://www.acq.osd.mil/evm/>。

产率参数为 5000， $B=0.37$)。假定该软件必须在 24 ± 12 个月的时间期限内交付。

- 25.7 假定你要为一所大学开发一个联机课程登记系统 (Online Course Registration System, OLCRS)。首先从客户的角度 (如果你是一名学生就很容易了!) 指出一个好系统应该具有的特性。(或者你的老师会为你提供一些初步的系统需求。) 按照第 24 章所介绍的估算方法, 估算 OLCRS 系统的开发工作量和工期。建议你按如下方式进行:
- a. 确定 OLCRS 项目中的并行工作活动。
 - b. 将工作量分布到整个项目中。
 - c. 建立项目里程碑。
- 25.8 为 OLCRS 项目选择适当的任务集。
- 25.9 为 25.7 题中的 OLCRS 或者你感兴趣的其他软件项目定义任务网络。确信你已给出所有的任务和里程碑, 并为每一项任务分配了所估算的工作量和工期。如果可能的话, 使用自动进度安排工具来完成这一工作。
- 25.10 如果有自动进度安排工具, 请为习题 25.9 中定义的任务网络确定关键路径。
- 25.11 使用进度安排工具 (如果有条件) 或者纸笔 (如果需要) 制定 OLCRS 项目的时序图。
- 25.12 假设你是一个软件项目管理者, 受命为一个小型软件项目进行挣值统计。这个项目共计划了 56 个工作任务, 估计需要 582 人日才能完成。目前有 12 个工作任务已经完成, 但是, 按照项目进度, 现在应该完成 15 个任务, 下面给出相关进度安排数据 (单位: 人日), 请你做出挣值分析。

任务	计划工作量	实际工作量	任务	计划工作量	实际工作量
1	12.0	12.5	9	12.0	10.0
2	15.0	11.0	10	6.0	6.5
3	13.0	17.0	11	5.0	4.0
4	8.0	9.5	12	14.0	14.5
5	9.5	9.0	13	16.0	—
6	18.0	19.0	14	6.0	—
7	10.0	10.0	15	8.0	—
8	4.0	4.5			

计算该项目的进度表执行指标 SPI、进度表偏差 SV、预定完成百分比、完成百分比、成本执行指标 CPI、成本偏差 CV。

扩展阅读与信息资源

事实上, 每一本关于软件项目管理的书都会包含进度安排的内容。项目管理研究所 (《PMBOK Guide》, 5th ed., PMI, 2013)、Wysoki (《Effective Project Management: Traditional, Agile, Extreme》, 6th ed., Wiley, 2011)、Lewis (《Project Planning Scheduling and Control》, 5th ed., McGraw-Hill, 2010)、Kerzner (《Project Management: A Systems Approach to Planning, Scheduling, and Controlling》, 10th ed., Wiley, 2009)、Chemuturi 和 Cagley (《Mastering Software Project Management: Best Practices, Tools, and Techniques》, J. Ross Publishing, 2010)、Hughes 和 Cotterel (《Software Project Management》, 5th ed., McGraw-Hill, 2009)、Luckey 和 Phillips (《Software Project Management for Dummies》, For Dummies, 2006)、Lewin (《Better Software Project Management》, Wiley, 2001) 以及 Bennatan (《On Time, Within Budget: Software Project Management Practices and Techniques》, 3rd ed., Wiley, 2000) 都包含了对这一主题的有价值的讨论。尽管 Harris (《Planning and

Scheduling Using Microsoft Office Project 2010》, Eastwood Harris Pty Ltd., 2010) 只是一个应用特例, 但是它深入探讨了如何有效地利用进度安排工具对软件项目进行跟踪和控制。

Fleming 和 Koppelman (《Earned Value Project Management》, 3rd edition, Project Management Institute Publications, 2010)、Budd (《A Practical Guide to Earned Value Project Management》, 2nd ed., Management Concepts, 2009) 以及 Webb 和 Wake (《Using Earned Value: A Project Manager's Guide》, Ashgate Publishing, 2003) 较详细地讨论了挣值技术在项目计划、跟踪和控制方面的应用。

网上有大量的与软件项目进度安排相关的信息资源。最新的参考文献参见 SEPA 网站 www.mhhe.com/pressman 下的 “software engineering resources”。

风险管理

要点浏览

概念: 很多问题都会困扰软件项目，风险分析和风险管理就是辅助软件团队理解和管理不确定事物的一系列步骤。风险是潜在的——它可能发生也可能不发生。但是，不管发生还是不发生，都应该去识别它，评估它发生的概率，估算它的影响，并制定它实际发生时的应急计划。

人员: 软件过程中涉及的每一个人——管理者、软件工程师和利益相关者——都要参与风险分析和风险管理。

重要性: 想想童子军的格言：“时刻准备着。”软件项目是困难重重的任务，很多事情都可能出错，而且坦率地说，很多事情经常出错。为此，时刻准备着——理解风险、采取主动的措施去回避或管理风险——是一个优秀的软件项目管理者应具备的基本条件。

步骤: 第一步称为“风险识别”，即辨别出什么情况下可能会出问题。第二步，分析每个风险，确定其可能发生的概率以及发生时将带来的危害。了解这些信息之后，就可以按照可能发生的概率和危害程度对风险进行排序。第三步，制定一个计划来管理那些发生概率高和危害程度大的风险。

工作产品: 风险缓解、监测和管理（Risk Mitigation, Monitoring and Management, RMMM）计划或一组风险信息表单。

质量保证措施: 所要分析和管理的风险，应该经过对人员、产品、过程和项目的彻底研究后再确定。RMMM 计划应该随着项目的进展而修订，以保证所考虑的风险是近期可能发生的。风险管理的应急计划应该是符合实际的。

Robert Charette[Cha89] 在他关于风险分析与管理的书中给出了风险概念的定义：

首先，风险涉及的是未来将要发生的事情。今天和昨天的事情已不再关心，如同我们已经在收获由我们过去的行为所播下的种子。问题是：我们是否能够通过改变今天的行为，而为一个不同的、充满希望的、更美好的明天创造机会。其次，风险涉及改变。如思想、观念、行为、地点的改变……第三，风险涉及选择，而选择本身就具有不确定性。因此，就像死亡和税收一样，风险是生活中最不确定的因素之一。

对于软件工程领域中的风险，Charette 的三条概念定义是显而易见的。未来是项目管理者所关心的——什么样的风险会导致软件项目彻底失败？改变也是项目管理者所关心的——客户需求、开发技术、目标环境以及所有其他与项目相关因素的改变将会对进度安排和总体成功产生什么影响？

关键概念

- 评估
- 识别
- 预测
- 细化
- 风险分类
- 风险显露度
- 风险条目检查表
- 风险表
- RMMM
- 安全和灾难
- 策略
 - 主动策略
 - 被动策略

最后,项目管理者必须抓住选择机会——应该采用什么方法及工具?需要多少人员参与?对质量的要求要达到什么程度才是“足够的”?

Peter Drucker [Dru75] 曾经说过:“当没有办法消除风险,甚至连试图降低该风险也存在疑问时,这个风险就是真正的风险了。”在弄清楚软件项目中的“真正风险”之前,识别出所有对管理者及开发者而言显而易见的风险是很重要的。

26.1 被动风险策略和主动风险策略

被动风险策略 (reactive risk strategy) 被戏称为“印第安纳·琼斯学派的风险管理” [Tho92]。印第安纳·琼斯在以其名字命名的电影中,每当面临无法克服的困难时,总是一成不变地说:“不要担心,我会想出办法来的!”印第安纳·琼斯从不担心任何问题,直到风险发生,再做出英雄式的反应。

引述 如果你不主动进攻风险,风险将会主动进攻你。

Tom Gilb

遗憾的是,一般的软件项目管理者并不是印第安纳·琼斯,软件项目团队的成员也不是他可信赖的伙伴。因此,大多数软件项目团队还是仅仅依赖于被动的风险策略。被动策略最多不过是针对可能发生的风险来监测项目,直到风险发生时,才会拨出资源来处理它们。大多数情况下,软件项目团队对风险不闻不问,直到出现了问题。这时,项目团队才赶紧采取行动,试图迅速地纠正错误,这通常叫作救火模式 (fire-fighting mode)。当这样的努力失败后,“危机管理” [Cha92] 接管一切,这时项目已经处于真正的危机中了。

对于风险管理,更好的是主动风险策略。主动 (proactive) 风险策略早在技术工作开始之前就已经启动了。识别出潜在的风险,评估它们发生的概率及产生的影响,并按其重要性进行排序。然后,软件项目团队就可以制定一个计划来管理风险。计划的主要目标是回避风险,但不是所有的风险都能够回避,所以,项目团队必须制定一个应急计划,使其在必要时能够以可控和有效的方式做出反应。在本章的后面将讨论风险管理的主动策略。

26.2 软件风险

虽然对于软件风险的严格定义还存在很多争议,但一般认为软件风险包含两个特性 [Hig95]: 不确定性 (uncertainty) 是指风险可能发生也可能不发生,即没有 100% 会发生的风险^①; 损失 (loss) 是指如果风险发生,就会产生恶性后果或损失。进行风险分析时,重要的是量化每个风险的不确定程度和损失程度。为了实现这一点,必须考虑不同类型的风险。

项目风险 (project risk) 威胁到项目计划。也就是说,如果项目风险发生,就有可能拖延项目的进度和增加项目的成本。项目风险是指预算、进度、人员 (聘用职员及组织)、资源、利益相关者、需求等方面的潜在问题以及它们对软件项目的影响。在第 24 章中,项目复杂度、规模及结构不确定性也属于项目 (和估算) 风险因素。

提问 在构建软件时可能会遇到什么类型的风险?

技术风险 (technical risk) 威胁到要开发软件的质量及交付时间。如果技术风险发生,开发工作就可能变得很困难或根本不可能。技术风险是指设计、实现、接口、验证和维护等方面的潜在问题。此外,规格说明的歧义性、技术的不确定性、技术陈旧以及“前沿”技术

① 100% 发生的风险是强加在软件项目上的约束。

也是技术风险因素。技术风险的发生是因为问题比我们所设想的更加难以解决。

商业风险 (business risk) 威胁到要开发软件的生存能力, 且常常会危害到项目或产品。五个主要的商业风险是: (1) 开发了一个没有人真正需要的优良产品或系统 (市场风险); (2) 开发的产品不再符合公司的整体商业策略 (策略风险); (3) 开发了一个销售部门不知道如何去销售的产品 (销售风险); (4) 由于重点的转移或人员的变动而失去了高级管理层的支持 (管理风险); (5) 没有得到预算或人员的保证 (预算风险)。

应该注意的是, 单一的风险分类并不总是行得通, 有些风险根本无法事先预测。

另一种常用的风险分类方式是由 Charette [Cha89] 提出的。已知风险 (known risk) 是通过仔细评估项目计划、开发项目的商业及技术环境以及其他可靠的信息来源 (如不现实的交付时间, 没有文档化需求或软件范围、恶劣的开发环境) 之后可以发现的那些风险。可预测风险 (predictable risk) 能够从过去项目的经验中推断出来 (如人员变动、与客户之间欠缺沟通、由于正在进行维护而使开发人员精力分散)。不可预测风险 (unpredictable risk) 就像纸牌中的大王, 它们可能会真的出现, 但很难事先识别。

引述 不经历实际风险的项目是不可能成功的。这种项目几乎是无益的, 否则早就有人开发了。

Tom DeMarco.

Tim Lister

信息栏 风险管理的 7 个原则

美国卡内基·梅隆大学软件工程研究所 (Software Engineering Institute, SEI, www.sei.cmu.edu) 定义了“实施有效风险管理框架”的 7 条原则:

保持全面观点——在软件所处的系统中考虑软件风险以及该软件所要解决的业务问题。

采用长远观点——考虑将来可能发生的风险 (如软件的变更), 并制定应急计划使将来发生的事件成为可管理的。

鼓励广泛交流——如果有人提出一个潜在的风险, 要重视它; 如果以非正式的方式提出一个风险, 要考虑它。任何时候都要

鼓励利益相关者和用户提出风险。

结合——考虑风险时必须与软件过程相结合。

强调持续的过程——在整个软件过程中, 团队必须保持警惕。随着信息量的增加, 要修改已识别的风险; 随着知识的积累, 要加入新的风险。

开发共享的产品——如果所有利益相关者共享相同版本的软件产品, 将更容易进行风险识别和评估。

鼓励协同工作——在风险管理活动中, 要汇聚所有利益相关者的智慧、技能和知识。

26.3 风险识别

风险识别试图系统化地指出对项目计划 (估算、进度、资源分配等) 的威胁。识别出已知风险和可预测风险后, 项目管理者首先要做的是在可能时回避这些风险, 在必要时控制这些风险。

26.2 节中提出的每一类风险又可以分为两种不同的类型: 一般风险和产品特定的风险。一般风险 (generic risk) 对每一个软件项目而言都是潜在的威胁。而产品特定的风险 (product-specific risk) 则只有那些对当前项目特定的技术、人员及环境非常了解的人才能识

别出来。为了识别产品特定的风险，必须检查项目计划及软件范围说明，然后回答这个问题：“本产品中有什么特殊的特性可能会威胁到我们的项目计划？”

识别风险的一种方法是建立风险条目检查表。该检查表可用于风险识别，并且主要用来识别下列几种类型中的一些已知风险和可预测风险：

- 产品规模 (product size) ——与要开发或要修改的软件的总体规模相关的风险。
- 商业影响 (business impact) ——与管理者或市场所施加的约束相关的风险。
- 项目相关人员特性 (stakeholder characteristic) ——与项目相关人员的素质以及开发者和项目相关人员定期沟通的能力相关的风险。
- 过程定义 (process definition) ——与软件过程定义的程度以及该过程被开发组织遵守的程度相关的风险。
- 开发环境 (development environment) ——与用来开发产品的工具的可得性及质量相关的风险。
- 开发技术 (technology to be built) ——与待开发软件的复杂性及系统所包含技术的“新奇性”相关的风险。
- 人员才干及经验 (staff size and experience) ——与软件工程师的总体技术水平及项目经验相关的风险。

风险条目检查表可以采用不同的方式来组织。与上述每个主题相关的问题可以针对每一个软件项目来回答。有了这些问题的答案，项目管理者就可以估计风险产生的影响。也可以采用另一种不同的风险条目检查表格式，即仅仅列出与每一种类型有关的特性。最终，给出一组“风险因素和驱动因子”[AFC88]以及它们发生的概率。有关性能、支持、成本及进度的驱动因子将在后面进行讨论。

网上有很多针对软件项目风险的检查表（例如，[Baa07]、[Nas07]、[Wor04]），项目管理者可以利用这些检查表来提高识别软件项目一般风险的洞察力。除了使用清单，风险模式[Mil04]也可作为系统的风险识别方法。

26.3.1 评估整体项目风险

下面的提问来源于对世界各地有经验的软件项目管理人员的调查而得到的风险资料[Kei98]，根据各个问题对项目成功的相对重要性将问题进行了排序。

1. 高层的软件管理者和客户管理者已经正式承诺支持该项目了吗？
2. 最终用户对项目和待开发的系统或产品热心支持吗？
3. 软件工程团队及其客户充分理解需求了吗？
4. 客户已经完全地参与到需求定义中了吗？
5. 最终用户的期望现实吗？
6. 项目范围稳定吗？
7. 软件工程团队的技能搭配合理吗？
8. 项目需求稳定吗？

建议 虽然考虑一般风险很重要，但是，通常产品特定的风险会带来更多的问题。一定要花时间尽可能多地识别出产品特定的风险。

提问 我们正在进行的软件项目面临严重的风险吗？

网络资源 风险雷达 (risk radar) 是一种数据库，也是一种工具，它可以帮助项目管理人员识别、排序和交流项目风险，见 www.spmn.com。

- 9. 项目团队对将实现的技术有经验吗?
- 10. 项目团队的人员数量满足项目需要吗?
- 11. 所有的客户或用户对项目的重要性和待开发的系统或产品的需求有共识吗?

如果对这些问题的任何一个回答是否定的, 则应务必启动缓解、监测和管理风险的步骤。项目的风险程度与对这些问题否定回答的数量成正比。

26.3.2 风险因素和驱动因子

美国空军有一本小册子 [AFC88], 其中包含了如何很好地识别和消除软件风险的指南。他们所用的方法是要求项目管理者识别影响软件风险因素的风险驱动因子, 风险因素包括: 性能、成本、支持和进度。在这里, 风险因素是以如下的方式定义的:

- 性能风险 (performance risk) ——产品能够满足需求且符合其使用目的的不确定程度。
- 成本风险 (cost risk) ——能够维持项目预算的不确定程度。
- 支持风险 (support risk) ——开发出的软件易于纠错、修改及升级的不确定程度。
- 进度风险 (schedule risk) ——能够维持项目进度且按时交付产品的不确定程度。每一个风险驱动因子对风险因素的影响均可分为四个影响类别: 可忽略的、轻微的、严重的或灾难的。图 26-1[Boe89] 指出了由于未识别出的软件失误而产生的潜在影响 (标号为 1 的行), 或没有达到预期的结果所产生的潜在影响 (标号为 2 的行)。影响类别的选择是最符合表中描述的特征为基础的。

引述 风险管理是针对成年人的项目管理。
Tim Lister

风险因素		性能	支持	成本	进度
影响类别					
灾难的	1	无法满足需求而导致任务失败		失误将导致进度延迟和成本增加, 预计超支 \$500K	
	2	严重退化使得根本无法达到要求的技术性能	无法做出响应或无法支持的软件	资金严重短缺, 很可能超出预算	无法在交付日期内完成
严重的	1	无法满足需求而导致系统性能下降, 使得任务能否成功受到质疑		失误将导致系统运行的延迟并使成本增加, 预计超支 \$100K 到 \$500K	
	2	技术性能有些降低	在软件修改中有少量的延迟	资金不足, 可能会超支	交付日期可能延迟
轻微的	1	无法满足需求而导致次要任务的降级		成本、影响和可以补救的进度延迟上的小问题, 预计超支 \$1K 或 \$100K	
	2	技术性能有些降低	较敏感的软件支持	有充足的资金来源	现实的、可完成的进度计划
可忽略的	1	无法满足需求而导致使用不方便或不易操作		失误对进度和成本的影响很小, 预计超支少于 \$1K	
	2	技术性能没有降低	易于进行软件支持	可以低于预算	交付日期将会提前

注: 1. 未识别出的软件失误或缺陷所产生的潜在影响。
2. 如果没有到达预期的结果所产生的潜在影响。

图 26-1 影响评估 [Boe89]

26.4 风险预测

风险预测 (risk projection) 又称风险估计 (risk estimation), 试图从两个方面评估每一个风险: (1) 风险发生的可能性或概率; (2) 如果风险发生, 风险相关问题产生的后果。项目计划人员、其他管理人员及技术人员都要进行以下 4 步风险预测活动:

- 1. 建立一个尺度, 以反映风险发生的可能性。
- 2. 描述风险产生的后果。
- 3. 估算风险对项目及产品的影响。
- 4. 标明风险预测的整体精确度, 以免产生误解。

按此步骤进行风险预测, 目的是使我们可以按照优先级来考虑风险。任何软件团队都不可能以同样的严格程度来为每个可能的风险分配资源, 通过将风险按优先级排序, 软件团队可以把资源分配给那些具有最大影响的风险。

26.4.1 建立风险表

风险表给项目管理者提供了一种简单的风险预测方法[⊖]。风险表样本如图 26-2 所示。

风险	类型	发生概率	影响值	RMMM
规模估算可能很不正确	PS	60%	2	
用户数量大大超出计划	PS	30%	3	
复用程度低于计划	PS	70%	2	
最终用户抵制该系统	BU	40%	3	
交付期限太紧	BU	50%	2	
资金将会流失	CU	40%	1	
用户将改变需求	PS	80%	2	
技术达不到预期的效果	TE	30%	1	
缺少对工具的培训	DE	80%	3	
人员缺乏经验	ST	30%	2	
人员变动比较频繁	ST	60%	2	
.....				

影响值:
1—灾难的 2—严重的 3—轻微的 4—可忽略的

图 26-2 排序前的风险表样本

项目管理者首先要在表中的第一列列出所有风险 (不管多么细微)。这可以利用 26.3 节所述的风险条目检查表来完成。在第二列中给出每一个风险的类型 (例如, PS 指产品规模风险, BU 指商业影响风险)。在第三列中填入各个风险发生的概率。各个风险的概率值可以首先由团队成员各自估算, 然后按循环投票的方式进行, 直到大家对风险概率的评估值趋于接近为止。

下一步是评估每个风险所产生的影响。使用图 26-1 所示的特性评估每

建议 尽力思考你将要开发的软件, 并问自己: “估计会出什么问题?” 建立你自己的列表, 并要求团队的其他成员也这么做。

⊖ 风险表可以采用电子表格模式来实现, 使表中的条目易于操作和排序。

个风险因素，并确定其影响类别。将4个风险因素——性能、支持、成本及进度——的影响类别求平均^①可得到一个整体的影响值。

完成了风险表的前4列内容之后，就可以按照概率和影响值来进行排序。高概率、高影响的风险放在表的上方，而低概率风险则移到表的下方。这样就完成了第一次风险排序。

项目管理者研究排序后的表，然后定义一条中截线。该中截线(cutoff-line，表中某处之上的一条水平线)表示：只有那些在中截线之上的风险才会得到进一步的关注，而在中截线之下的风险则需要重新评估以进行第二次排序。参照图26-3，从管理关注的角度来看，风险的影响和发生的概率是截然不同的。一个具有高影响但发生概率很低的风险因素不应该耗费太多的管理时间，而高影响且发生概率为中到高的风险，以及低影响且高概率的风险，则应该首先列入随后的风险分析步骤中。

所有在中截线之上的风险都必须进行管理。标有RMMM的列中包含了一个指示器，指向为所有中截线之上的风险所建立的风险缓解、监测及管理计划(Risk Mitigation, Monitoring and Management Plan, RMMM计划)或一组风险信息表单。RMMM计划和风险信息表单将在26.7节讨论。

风险概率可以通过先做个别估计而后求出一个有代表性的值来确定。虽然这个方法是可行的，不过还有很多其他更复杂的确定风险概率的技术(如[McC09])。

26.4.2 评估风险影响

如果风险真的发生了，那么有三个因素可能会影响风险所产生的后果，即风险的本质、范围和时间。风险的本质是指当风险发生时可能带来的问题。例如，一个定义很差的与客户硬件的外部接口(技术风险)会妨碍早期的设计和测试，也有可能项目后期阶段的系统集成问题。风险的范围包括风险的严重性(即风险有多严重)及风险的整体分布情况(项目中有多少部分受到影响或有多少客户受到损害)。最后，风险的时间是指何时能够感受到风险的影响及风险的影响会持续多长时间。在大多数情况下，项目管理者希望“坏消息”越早出现越好，但在某些情况下则是越迟越好。

让我们再回到美国空军提出的风险分析方法[AFC88]上来。下面的步骤可以用来确定风险的整体影响：(1)确定每个风险因素发生的平均概率。(2)使用图26-1中列出的标准来确定每个因素的影响。(3)按照前面几节给出的方法填写风险表，并分析其结果。

关键点 在风险表中应该按照概率和影响值对风险进行排序。

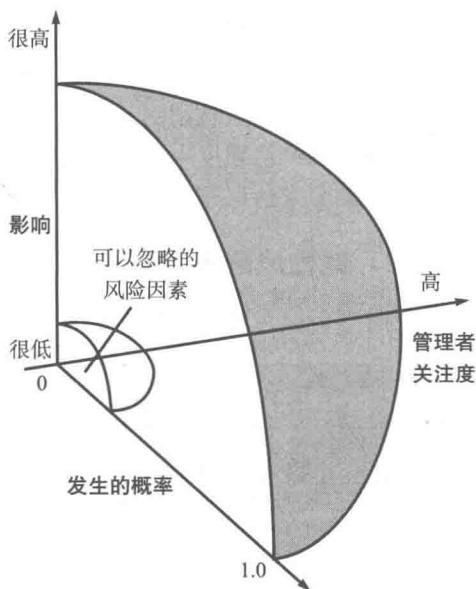


图 26-3 风险与管理

引述 如今，谁也不会奢望能够很好地了解任务以使其不会使人感到惊讶，而惊讶就意味着风险。

Stephen Grey

① 如果某个风险因素对项目来说比较重要，则可以使用加权平均法。

整体的风险显露度 (Risk Exposure, RE) 可由下面的关系确定 [HAL98]:

$$RE = P \times C$$

其中 P 是风险发生的概率, C 是风险发生时带来的项目成本。

例如, 假设软件团队按如下方式定义了项目风险。

风险识别: 事实上, 计划可复用的软件构件中只有 70% 将集成到应用中, 其他功能必须定制开发。

风险概率: 80% (大约)。

风险影响: 计划了 60 个可复用的软件构件, 如果只能利用 70%, 则 18 个构件必须从头开发 (除了已经计划开发的定制软件外)。平均每个构件的程序行数是 100 LOC, 本地数据表明每个 LOC 的软件工程成本是 \$14.00, 开发构件的总成本 (影响) 将是 $18 \times 100 \times 14 = \25200 。

风险显露度: $RE = 0.80 \times \$25200 \approx \20200

风险的成本估算完成之后, 就可以为风险表中的每个风险计算其风险显露度。所有风险 (风险表中截线之上) 的总体风险显露度不仅为调整项目最终的成本估算提供了依据, 还可预测在项目进展过程中不同阶段所需人员资源的增长情况。

26.4.1 节和 26.4.2 节所述的风险预测和分析方法可以在软件项目进展过程中反复运用^①。项目团队应该定期复查风险表, 重新评估每一个风险, 以确定新环境是否引起其概率和影响发生改变。这样可能需要在风险表中添加一些新的风险, 删除一些不再有影响的风险, 或改变一些风险的相对位置。

提问 如何评估风险产生的后果?

建议 将所有风险的 RE 与项目的成本估算进行比较, 如果 RE 大于项目成本的 50%, 则必须重新评估项目的生存能力。

SafeHome 风险分析

[场景] Doug Miller 的办公室, SafeHome 软件项目开始之前。

[人物] Doug Miller (SafeHome 软件团队经理)、Vinod Raman、Jamie Lazar 以及产品软件工程团队的其他成员。

[对话]

Doug: 很高兴今天和大家一起讨论 SafeHome 项目的风险问题。

Jamie: 是讨论什么情况下可能会出问题吗?

Doug: 是的。这儿有几种可能会出问题的类型。(他给每个人展示了 26.3 节中给出的类型。)

Vinod: 嗯……你只是要求我们找出风险,

还是……

Doug: 不, 我想让每一个人建立一个风险列表, 立即动手……

(10 分钟过去了, 每个人都在写着。)

Doug: 好了, 停下来。

Jamie: 可是我还没有完成!

Doug: 没关系, 我们还要对列表进行复查。现在, 给列表中的每一个风险指定其发生概率的百分比值, 然后按 1 (较小的) 到 5 (灾难的) 的取值范围确定其对项目的影响。

Vinod: 就是说如果我认为风险的发生跟掷硬币差不多, 就给 50% 的概率, 如果我认为风险的影响是中等的, 就给影响

^① 如果你有兴趣, 可阅读 [Ben10] 中给出的风险成本数学处理的相关内容。

值为 3，对吗？

Doug：非常正确。

（5 分钟过去了，每个人都在写着。）

Doug：好了，停下来。现在，我们在白板上建立一组列表，我来写，轮流从你们各自的列表中取出一项。

（15 分钟过去了，列表完成。）

Jamie（指着白板并笑着说）：Vinod，那个风险（指向白板中的一项）很可笑，大家意外选中它的可能性很大，应该删除。

Doug：不，先留着吧。不管有多么不可思议，我们应考虑所有风险。一会儿我们还要精简这个列表。

Jamie：已经有 40 多个风险了，我们究竟怎样才能管理它们呢？

Doug：管理不了。所以将这些风险排序之后，我们还要定义中截线。明天我们继续开会讨论中截线。现在，回去继续工作，工作之余考虑是否还有遗漏的风险。

26.5 风险细化

在项目计划的早期，风险很可能只是一个大概的描述。随着时间的推移，我们对项目和风险的了解加深，可以将风险细化为一组更详细的风险，在某种程度上，这些风险更易于缓解、监测和管理。

实现方法之一是按条件-转化-结果（Condition-Transition-Consequence, CTC）格式 [GLU94] 来表示风险，即采用如下方式来描述风险：

提问 用什么好方式来描述风险？

给定 < 条件 >，则（可能）导致 < 结论 >

使用 CTC 格式，在 26.4.2 节中提到的可复用软件构件的风险可描述为：

给定条件：所有可复用软件构件必须符合特定设计标准，但是某些并不符合。则有结论：（可能）仅 70% 的计划可复用构件将集成到最终的系统中，需定制开发剩余 30% 的构件。

可按如下方式对这个一般条件进行细化：

子条件 1。某些可复用构件是由第三方开发的，没有其内部设计标准的相关资料。

子条件 2。构件接口的设计标准尚未确定，有可能和某些现有的软件可复用构件不一致。

子条件 3。某些可复用构件是采用不支持目标环境的语言开发的。

这些子条件的结论是相同的（即必须定制开发 30% 的软件构件），但细化过程可以帮助我们排除重大风险，使我们更易于分析风险和采取措施。

26.6 风险缓解、监测和管理

这里讨论的所有风险分析活动只有一个目的——辅助项目团队制定处理风险的策略。一个有效的策略必须考虑三个问题：风险回避、风险监测、风险管理及应急计划。

如果软件团队采取主动的方法，最好的策略就是风险回避。这可以通过建立一个风险缓解（risk mitigation）计划来实现。例如，假设频繁的人员变动被标注为项目风险 r_1 。基于以往的历史和管理经验，可以估计频繁人员变动的概率 I_1 为 0.70（70%，相当高），并预测影响 x_1 为严重的。也就是说，频繁的人员变动将对项目成本及进度有严重的影响。

引述 我采取如此多的预防措施，是因为我不想冒任何风险。

Napoleon

为了缓解这个风险，项目管理者必须制定一个策略来减少人员变动。可能采取的步骤包括：

- 与现有人员一起探讨人员变动的起因（如恶劣的工作条件、报酬低、竞争激烈的劳动力市场）。
提问 如何缓解风险？
- 在项目开始之前采取行动，设法缓解那些我们能够控制的起因。
- 项目启动之后，假设会发生人员变动，当人员离开时，找到能够保证工作连续性的方法。
- 组织项目团队，使得每一个开发活动的信息能被广泛传播和交流。
- 制定工作产品标准，并建立相应机制以确保能够及时创建所有的模型和文档。
- 同等对待所有工作的评审（使得不止一个人能够“跟上进度”）。
- 给每一个关键的技术人员都指定一个后备人员。

随着项目的进展，风险监测（risk-monitoring）活动开始了，项目管理者应该监测那些可以表明风险是否正在变高或变低的因素。在人员变动频繁的例子中，应该监测：团队成员对项目压力的普遍态度、团队的凝聚力、团队成员彼此之间的关系、与报酬和利益相关的潜在问题、在公司内及公司外工作的可能性。

除了监测上述因素之外，项目管理者还应该监测风险缓解步骤的效力。例如，前面叙述的风险缓解步骤中要求制定工作产品标准，并建立相应机制以确保能够适时开发出工作产品。万一有关键成员离开此项目，应该有一个保证工作连续性的机制。项目管理者应该仔细监测这些工作产品，以保证每一个工作产品的正确性，在项目进行中有新员工加入时，能为他们提供必要的信息。

风险管理及应急计划（risk management and contingency planning）是以缓解工作已经失败而且风险已经发生为先决条件的。继续前面的例子，假定项目正在进行之中，有一些人宣布将要离开。如果已经按照缓解策略行事，则有后备人员可用，信息已经文档化，有关知识已经在团队中广泛进行了交流。此外，对那些人员充足的岗位，项目管理者还可以暂时重新调整资源（并重新调整项目进度），从而使得新加入团队的人员能够“赶上进度”。同时，应该要求那些将要离开的人员停止所有的工作，并在最后几星期进入“知识交接模式”。比如，准备录制视频知识，建立“注释文档或 Wiki”，或者与仍留在团队中的成员进行交流。

值得注意的是，RMMM 步骤会导致额外的项目成本。例如，花时间给每个关键技术人员配备“后备人员”得承担费用。因此，风险管理的另一个任务就是评估什么情况下由 RMMM 步骤所产生的效益高于实现这些步骤所需的成本。通常，项目管理者要进行典型的成本/效益分析。如果频繁的人员变动风险的缓解步骤经评估将会增加 15% 的项目成本和工期，而主要的成本因素是“配备后备人员”，则管理者可能决定不执行这一步骤。另一方面，如果风险缓解步骤经预测仅增加 5% 的项目成本和 3% 的工期，则管理者极有可能将这一步骤付诸实现。

建议 如果某特定风险的 RE 小于其风险缓解的成本，则不用试图缓解该风险，而是继续监测。

对于大型项目，可以识别出 30 或 40 种风险。如果为每一个风险制定 3 ~ 7 个风险缓解步骤，则风险管理本身就可能变成一个“项目”！因此，项目管理者可以将 Pareto 的 80-20 法则用于软件风险上。经验表明，整个项目 80% 的风险（即可能导致项目失败的 80% 的潜在因素）可能是由只占 20% 的已经识别出的风险所引发。早期风险分析步骤中所做的工作能够帮助项目管理者确定哪些风险在这 20% 中（如导致高风险显露度的风险）。因此，某些

已经识别、评估和预测过的风险可能并不被纳入 RMMM 计划之中——这些风险不属于那关键的 20%（具有最高项目优先级的风险）。

风险并不仅限于软件项目本身。在软件已经成功开发并交付给客户之后，仍有可能发生风险。这些风险一般与该领域中的软件缺陷相关。

软件安全和灾难分析（software safety and hazard analysis，例如 [Dun02]、[Her00]、[Lev95]）是一种软件质量保证活动（第 16 章），主要用来识别和评估可能对软件产生负面影响并促使整个系统失效的潜在灾难。如果能够在软件工程过程的早期阶段识别灾难，就可以使用某些软件设计特性来消除或控制这些潜在的灾难。

26.7 RMMM 计划

风险管理策略可以包含在软件项目计划中，也可以将风险管理步骤组织成一个独立的风险缓解、监测和管理计划（RMMM 计划）。RMMM 计划将所有风险分析工作文档化，项目管理者还可将其作为整个项目计划的一部分。

有些软件团队并不建立正式的 RMMM 文档，而是将每个风险分别使用风险信息表单（Risk Information Sheet, RIS）[Wil97] 进行文档化。在大多数情况下，RIS 采用数据库系统进行维护，这样容易完成创建、信息输入、优先级排序、查找以及其他分析。RIS 的格式如图 26-4 所示。

风险信息表单			
风险标识号：P02-4-32	日期：5/9/09	概率：80%	影响：高
描述 事实上，计划可复用的软件构件中只有 70% 将集成到应用中，其他功能必须定制开发。			
细化 / 环境 子条件 1：某些可复用构件是由第三方开发的，没有其内部设计标准的相关资料。 子条件 2：构件接口的设计标准尚未确定，有可能和某些现有的软件可复用构件不一致。 子条件 3：某些可复用构件是采用不支持目标环境的语言开发的。			
缓解 / 监测 1. 与第三方交流以确定其与设计标准的符合程度。 2. 强调接口标准的完整性，在确定接口协议时应考虑构件的结构。 3. 检查并确定属于子条件 3 的构件数量，检查并确定是否能够获得语言支持。			
管理 / 应急计划 / 触发 RE 的计算结果为 \$20200。在项目应急计划中分配这些费用。修订进度表，假定必须定制开发 18 个附加构件。据此分配人员。 触发：缓解步骤自 7/1/09 起没有效果。			
当前状态 5/12/09：缓解步骤启动。			
创建者：D. Gagne		受托者：B. Laster	

图 26-4 风险信息表单 [Wil97]

建立了 RMMM 计划，而且项目已经启动之后，风险缓解及监测步骤也就开始了。正如前面讨论过的，风险缓解是一种问题规避活动，而风险监测则是一种项目跟踪活动，这种监测活动有三个主要目的：（1）评估所预测的风险是否真正发生了；（2）保证正确地实施了各

风险的缓解步骤；(3) 收集能够用于今后风险分析的信息。在很多情况下，项目中发生的问题可以追溯到不止一个风险，所以风险监测的另一个任务就是试图找到“起源”(在整个项目中是哪些风险引起了哪些问题)。

软件工具 风险管理

[目标] 风险管理工具的目的是辅助项目团队识别风险，评估风险的影响及发生的概率，并在整个软件项目过程中跟踪风险。

[机制] 通常，风险管理工具能够提供典型的项目和商业风险列表来辅助识别一般风险；能够提供检查表或其他“接口”技术来辅助识别项目特定的风险；能够指定每一个风险发生的概率及影响；支持风险缓解策略；能够生成多种不同的风险相关报告。

[代表性工具]^①

- **@Risk**。由 Palisade Corporation (www.palisade.com) 开发，是一个利用蒙特卡罗 (Monte Carlo) 模拟法来驱动分析机的一般风险分析工具。
- **Riskman**。由 ABS Consulting (www.abs-consulting.com/riskmansoftware/index.html) 发布，是能够识别项目相关风险的一个风险评估专家系统。
- **Risk Radar**。由 SPMN (www.spmn.com) 开发，能够辅助项目管理者识别和管理项目风险。
- **ARM**。由 Deltek (www.deltek.com) 开发，这是一个基于 Web 的工具，可使供应商、客户及地处异地的项目团队共享必要的风险认知。
- **X:PRIMER**。由 GrafP Technologies (www.grafp.com) 开发，是一个通用的 Web 工具，可预测项目中什么可能出错，并能够识别出潜在错误的根本原因并制定有效的应对措施。

习题与思考题

- 26.1 举出 5 个其他领域的例子来说明与被动风险策略相关的问题。
- 26.2 简述“已知风险”和“可预测风险”之间的差别。
- 26.3 为 SEPA 站点上给出的每个风险条目检查表增加三个问题或主题。
- 26.4 你受命开发一个支持低成本的视频编辑系统的软件。该系统输入数字视频信号，将视频信息存在磁盘上，然后允许用户对数字化的视频信息进行各种编辑，最后生成 DVD 或其他多媒体格式。对这类系统做一些调研，然后列出当你开始启动这类项目时，将面临的技术风险是什么。
- 26.5 假如你是某大型软件公司的项目经理，且受命领导一个团队开发下一代字处理软件，请给该项目提供一个风险表。
- 26.6 说明风险因素和风险驱动因子的差别。
- 26.7 为图 26-2 所描述的三个风险制定风险缓解策略及特定的风险缓解活动。
- 26.8 为图 26-2 所描述的三个风险制定风险监测策略及特定的风险监测活动。确保你所监测的风险因素可以确定风险正在变大或变小。
- 26.9 为图 26-2 所描述的三个风险开发风险管理策略和特定的风险管理活动。
- 26.10 细化图 26-2 中的三个风险，并为每个风险建立风险信息表单。
- 26.11 用 CTC 格式表示图 26-2 中的三个风险。
- 26.12 假定每个 LOC 的成本为 \$16，概率为 60%，重新计算 26.4.2 节中讨论的风险显露度。

26.13 你能否想到一种情况：一个高概率、高影响的风险并未纳入 RMMM 计划的考虑之中？

26.14 给出主要关注软件安全和灾难分析的 5 个软件应用领域。

扩展阅读与信息资源

近几十年来，软件风险管理方面的文献已有很多。Mun (《Modeling Risk》，2nd ed., Wiley, 2010) 详细介绍了适用于软件项目的风险分析数学处理方法。Mulcahy (《Risk Management, Tricks of the Trade for Project Managers》2nd ed., RMC Publications, 2010)、Kendrick (《Identifying and Managing Project Risk》2nd ed., American Management Association, 2009)、Crohy 及其同事 (《The Essentials of Risk Management》，McGraw-Hill, 2006) 以及 Marrison (《The Fundamentals of Risk Measurement》，McGraw-Hill, 2002) 介绍了每个项目采用的各种有用的方法及工具。Jindal 及其同事 (《Risk Management in Software Engineering, Create Space in Dependent Publishing》，2012) 讨论了作为系统开发一部分的嵌入式安全风险评估。

DeMarco 和 Lister 写了一本有趣而意义深刻的书 (《Dancing with Bears》，Dorset House, 2003)，该书可以指导软件管理者和开发者进行风险管理。Moynihan (《Coping with IT/IS Risk Management》，Springer-Verlag, 2002) 介绍了项目风险管理者所采用的实用方法。Royer (《Project Risk Management》，Management Concepts, 2002) 以及 Smith 和 Merritt (《Proactive Risk Management》，Productivity Press, 2002) 论述了项目管理的主动过程。Karolak 撰写了一本参考书 (《Software Engineering Risk Management》，Wiley, 2002)，书中介绍了一种便于使用的风险分析模型，以及一些有价值的检查表和调查表软件包。

Capers Jones (《Assessment and Control of Software Risks》，Prentice-Hall, 1994) 对软件风险进行了详细讨论，其中包含从数百个软件项目中收集的数据，Jones 定义了 60 个可能影响软件项目结果的风险因素。Boehm[Boe89] 给出了很好的调查表和检查表格式，对风险识别具有重大作用。Charette[Cha89] 描述了风险分析方法的详细处理，采用了概率论和统计技术来分析风险。Charette 的另一本书 (《Application Strategies for Risk Analysis》，McGraw-Hill, 1990) 从系统和软件工程的角度讨论风险，并提出了实用的风险管理策略。Gilb (《Principles of Software Engineering Management》，Addison-Wesley, 1988) 提出了一组“原则” (通常是有趣的，有时是深刻的)，可作为风险管理的有效指南。

Ewusi-Mensah (《Software Development Failures : Anatomy of Abandoned Projects》，MIT Press, 2003) 和 Yourdon (《Death March》，Prentice-Hall, 1997) 讨论了当灾难性软件项目风险发生时会给软件团队带来什么后果。Bernstein (《Against the Gods》，Wiley, 1998) 描述了有趣的远古时期的风险历史。

美国卡内基·梅隆大学软件工程研究所 (SEI) 已经出版了很多关于风险分析和风险管理的详细报告和参考书。美国空军司令部手册 AFSCP 800-45[AFC88] 描述了风险识别及降低技术。《ACM Software Engineering Notes》每期都有题为“Risk to the Public”(编辑：P. G. Neumann) 的讨论，如果你想知道最新和最好的软件恐怖故事，这里就有。

网上有大量与软件风险管理相关的信息资源，最新的风险管理参考文献见 SEPA 网站 www.mhhe.com/pressman 上的“software engineering resource”专题。

[General Information]

书名=软件工程 实践者的研究方法 原书第8版 本科教学版

作者=(美) 罗杰·S.普莱斯曼 (Roger

页数=394

SS号=14157763

DX号=

出版日期=2017.01

出版社=北京机械工业出版社